COMPARAÇÃO SISTEMÁTICA DO DESEMPENHO DE CLASSIFICADORES BINÁRIOS SUPERVISIONADOS

Matheus Avellar de Barros

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientadores: Carlos Eduardo Pedreira
Laura de Oliveira F. Moraes

Rio de Janeiro
Maio de 2025

COMPARAÇÃO SISTEMÁTICA DO DESEMPENHO DE CLASSIFICADORES BINÁRIOS SUPERVISIONADOS

Matheus Avellar de Barros

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientadores: Carlos Eduardo Pedreira
Laura de Oliveira Fernandes Moraes

Aprovada por: Prof. Carlos Eduardo Pedreira
Prof.ª Laura de Oliveira Fernandes Moraes
Prof.ª Carolina Gil Marcelino
Prof. Carlos Eduardo Ribeiro de Mello

RIO DE JANEIRO, RJ – BRASIL
MAIO DE 2025

Barros, Matheus Avellar de

Comparação Sistemática do Desempenho de Classificadores Binários Supervisionados/Matheus Avellar de Barros. – Rio de Janeiro: UFRJ/COPPE, 2025.

XIV, 85 p.: il.; 29, 7cm.

Orientadores: Carlos Eduardo Pedreira

Laura de Oliveira Fernandes Moraes

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2025.

Referências Bibliográficas: p. 74 – 80.

1. Classificação. 2. Classificador supervisionado. 3. Classificador binário. 4. Geração de dados. 5. Otimização de parâmetros. I. Pedreira, Carlos Eduardo *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Everything in the world is good. But if you're not careful, even good things can turn against you.

—YOSHIKAWA, E. "Taiko: An Epic Novel of War and Glory in Feudal Japan". Tradução: William Scott Wilson. Kodansha International, 1992. p. 130.

Agradecimentos

Gostaria de agradecer, primeiramente, ao meu orientador, prof. Pedreira, por me acolher quando precisei realizar a troca de área de pesquisa. Mesmo que tenha sido um processo inerentemente caótico, ele foi paciente na minha adaptação para uma área na qual eu tinha pouco conhecimento.

Além disso, preciso estender minha gratidão, em igual quantidade, à minha orientadora, prof.ª Laura. Sem seu apoio, sugestões, comentários e ideias durante todo o desenvolvimento do projeto, este trabalho não teria sido possível.

Também não posso deixar de agradecer ao prof. Claudio Esperança, pela ajuda que me ofereceu no período de transição de áreas de pesquisa. Sua preocupação e empatia definitivamente tornaram o processo de troca muito mais simples.

Finalmente, agradeço ao Gutierrez da Costa, pelo trabalho fenomenal e indispensável na secretaria acadêmica; ao Ricardo Cézar Vieira da Silva Júnior, por seu grande suporte nas minuciosidades burocráticas; e ao CNPq, pela oportunidade em forma de bolsa de fomento à pesquisa.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

COMPARAÇÃO SISTEMÁTICA DO DESEMPENHO DE CLASSIFICADORES BINÁRIOS SUPERVISIONADOS

Matheus Avellar de Barros

Maio/2025

Orientadores: Carlos Eduardo Pedreira
Laura de Oliveira Fernandes Moraes

Programa: Engenharia de Sistemas e Computação

Algoritmos de classificação, ou classificadores, categorizam dados em classes distintas, e cumprem um papel fundamental em aprendizado de máquina. Classificadores são usados em diversas áreas, para a detecção de *spam* e fraudes bancárias, o reconhecimento de objetos em imagens (como pessoas, animais e gestos) e até a detecção de células cancerígenas. Como a tarefa de classificação não é trivial, ou mesmo determinística, é necessária a implementação de heurísticas para realizar a tomada de decisão em casos ambíguos – e essa é a função dos algoritmos. Contudo, diante a uma ampla gama de opções, as diferenças pragmáticas entre os diversos classificadores não são sempre claras. Nesta dissertação, mostra-se que é possível agrupar certos classificadores binários supervisionados amplamente utilizados na literatura (SVM, GBDT, kNN, RF e NB) conforme sua resistência a dados com ruído e/ou desequilibrados, ou quanto a seu tempo de execução. Esses resultados indicam que, mesmo que haja grande similaridade no desempenho geral de algoritmos bem conhecidos, eles possuem diferenças de execução que podem ter um impacto significativo a depender do conjunto de dados sendo trabalhado. Assim, os resultados podem auxiliar na escolha por um classificador mais adequado.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

SYSTEMATIC COMPARISON OF SUPERVISED BINARY CLASSIFIERS' PERFORMANCE

Matheus Avellar de Barros

May/2025

Advisors: Carlos Eduardo Pedreira
Laura de Oliveira Fernandes Moraes

Department: Systems Engineering and Computer Science

Classification algorithms, or classifiers, categorize data into distinct classes, and play a fundamental role in machine learning. Classifiers are used in various areas, from detecting *spam* and financial fraud, to recognizing objects in images (such as people, animals and gestures) and even detecting cancer cells. Because the classification task isn't trivial, or even deterministic, implementing heuristics to aid in the decision-making process for ambiguous cases is necessary – and that's the algorithms' function. However, faced with a plethora of options, pragmatic differences between the many classifiers are not always clear. In this dissertation, we show that it is possible to group certain supervised binary classifiers widely used in the literature (SVM, GBDT, kNN, RF and NB) according to their resistance to noise and/or unbalanced data, or to their running time. These results indicate that, even if there is a great deal of similarity in the general performance of well known algorithms, they have differences in execution that may have a significant impact depending on the dataset being used. Therefore, the results may assist in the choice for a more adequate classifier.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

AB	AdaBoost 8
AUC	Area Under ROC Curve 7
BNN	Backpropagation Neural Network 56
CART	Classification And Regression Tree 8
DL	Deep Learning 7
DT	Decision Tree 56
ELM	Extreme Learning Machine 7
GBDT	Stochastic Gradient Boosted Decision Trees 4
kNN	k-Nearest Neighbors 4
LogR	Logistic Regression 8
MSE	Mean Squared Error 13
NB	Gaussian Naïve Bayes 4
PT	Parameter Tuning 36
RBFNN	RBF Neural Network 56
RF	Random Forest 4
RMSE	Root Mean Squared Error 34
SAR	Squared error, Accuracy and ROC area 34
SRC	Sparse Representation-based Classification 8

SVM	Support Vector Machines 3
SVM pol.	SVM com *kernel* polinomial 25
SVM pré-comp.	SVM com *kernel* pré-computado 25
SVM RBF	SVM com *kernel* gaussiano 12
UCI	University of California, Irvine 7

Capítulo 1

Introdução

Desde os princípios da Internet, avanços tecnológicos têm permitido uma cada vez maior geração de dados por seus usuários. No início do século, a sociedade gradualmente entrou no que é conhecido como a "Web 2.0". Aqui, o uso de Internet migra de uma fase inicial exploratória, conhecida como "Web 1.0", para uma de produção de conteúdo pessoal e o compartilhamento desse conteúdo em plataformas online (BLANK e REISDORF, 2012). Inevitavelmente, essa mudança trouxe consigo um crescimento explosivo na geração e publicação de dados.

Um exemplo bastante ilustrativo é o caso do YouTube, plataforma de compartilhamento de vídeos fundada em 2005.[1] MCGRADY *et al.* (2023), em um estudo sobre a plataforma, estimam que ela possuía, ao final de 2022, aproximadamente 10 bilhões de vídeos públicos – mais do que um vídeo por pessoa do planeta – além de uma quantidade desconhecida de vídeos não-listados e privados. Ademais, os autores apontam que a submissão de novos vídeos ainda está em uma fase de crescimento: quase metade (46%) dos vídeos encontrados havia sido submetida entre 2021 e 2022.

Isso é indicativo de uma tendência maior. Conforme mais espaços do cotidiano migram para o digital – seja na forma de *smartphones*, *smartwatches*, *tablets*, fones de ouvido inteligentes, etc – mais e mais dados são constantemente gerados e, frequentemente, publicados. E, em muitas ocasiões, é necessário reconhecer padrões nos dados para, por exemplo, poder separá-los em categorias distintas. É o caso do Google Photos, plataforma de *backup* de fotografias em nuvem. Por trás dos panos, o programa usa tecnologias de aprendizado de máquina para reconhecer pessoas, animais e objetos nas fotografias, permitindo que estes sejam buscados textualmente posteriormente.[2]

Contudo, devido ao grande volume de dados produzidos, essa classificação não

[1] Disponível em: `https://washingtonpost.com/wp-dyn/content/article/2006/10/06/AR2006100600660.html`. Acesso em 22 dez.2024.

[2] Disponível em: `https://mashable.com/article/google-photos-auto-white-balance`. Acesso em 22 dez.2024.

pode ser feita manualmente. Assim, métodos automatizados para realizar classificação de dados foram desenvolvidos, chamados convencionalmente de "classificadores". Dentre as diversas áreas que fazem uso da classificação de dados, destaca-se: a detecção de *spam* e fraudes bancárias (ABDALLAH *et al.*, 2016; KARIM *et al.*, 2019; SAIDANI *et al.*, 2020); o reconhecimento de objetos, como pessoas, animais ou gestos, em imagens (GUTTA e WECHSLER, 1996; HORIGUCHI *et al.*, 2018); a inferência de emoções a partir de texto (ELGELDAWI *et al.*, 2021); e até a detecção de células cancerígenas (KARABATAK, 2015; REJANI e SELVI, 2009).

Um classificador, de modo geral, é um algoritmo de aprendizado de máquina que realiza o reconhecimento de padrões para discernir a quais grupos (ou "classes") pertence um conjunto de observações. Porém, esses algoritmos não são todos iguais; para cada problema, existe um tipo de classificação adequada.

Este trabalho restringe-se a comparar classificadores *binários supervisionados*. O classificador é dito *binário* quando categoriza observações em somente duas classes, "sim" e "não". Por sua vez, um método é *multiclasse* se não possui tal restrição.

Um classificador dito *supervisionado* conhece, *a priori*, a quais classes os dados podem pertencer, em contraste a um classificador *não-supervisionado*, que não possui informação prévia sobre a classificação dos dados (ABU-MOSTAFA *et al.*, 2012). Por exemplo, um classificador supervisionado pode ser instruído a classificar emails em "*spam*" e "não *spam*", e receber exemplos de cada um para auxiliar na tomada de decisão: emails com links para certos websites, oferecendo dinheiro, ou com algumas palavras-chave serão mais provavelmente classificados como *spam* (KARIM *et al.*, 2019). Por outro lado, um classificador não-supervisionado pode receber um conjunto de dados contendo medidas de circunferência e massa de moedas, e ele conseguirá separá-las em quantas classes forem necessárias, mesmo sem conhecimento do que cada classe representa (ABU-MOSTAFA, 2012).

Além disso, o mérito de um classificador não está somente relacionado à sua capacidade de classificar corretamente. Há um fator adicional a ser considerado: a capacidade de *generalização* do problema. Um algoritmo que perfeitamente classifica dados de treino está, na realidade, aprendendo ruído; foi treinado para classificar somente os dados que já recebeu, mas nada além. Essa situação recebe o nome de *overfitting* (ABU-MOSTAFA *et al.*, 2012). Nesse caso, deparando-se com dados novos, tal algoritmo terá pior desempenho do que um mais equilibrado, que reconhece situações ruidosas e mantém um limiar de decisão mais próximo ao do processo que está gerando os dados (ABU-MOSTAFA *et al.*, 2012).

Assim, frequentemente, prefere-se um algoritmo que classifica dados de treino imperfeitamente, com a intenção de que ele aprenda uma generalização do problema e seja bem-sucedido com dados inéditos (DUDA *et al.*, 2000).

Em um panorama tão complexo, há muitas considerações a serem feitas antes

de decidir pelo uso de um algoritmo específico. E, mesmo se tratando somente de classificadores binários supervisionados, ainda existe uma ampla gama de opções; alguns nomes populares incluem árvore de decisão, vizinhos mais próximos, *Support Vector Machines* (SVM), entre outros (FERNÁNDEZ-DELGADO *et al.*, 2014).

Crucialmente, não há um único algoritmo "superior", para ser usado sempre. Ainda que todas as condições dos dados estejam perfeitas – em termos, por exemplo, de ruído e desequilíbrio de dados – cada método possui vantagens e desvantagens inerentes a seu funcionamento que vão, inevitavelmente, variar o desempenho deste em diferentes situações. Dessa maneira, um pesquisador, quando deparado com a necessidade de aplicar o reconhecimento de padrões em dados, pode acabar tendo dificuldades em determinar qual classificador deve usar.

No intuito de auxiliar essa tomada de decisão e evidenciar as vantagens e desvantagens de diversos classificadores, há uma abundância de estudos anteriores que se propuseram a compará-los (AMANCIO *et al.*, 2014; FERNÁNDEZ-DELGADO *et al.*, 2014; LIM *et al.*, 2000; SINGH *et al.*, 2017; ZHANG *et al.*, 2017). Entretanto, dois grandes problemas foram encontrados nesses trabalhos – explorados em mais detalhes no capítulo 2.

Primeiramente, há uma tendência em usar somente dados reais, e nunca sintéticos, de forma a demonstrar a utilidade dos algoritmos na vida real. Ainda que isso possa ocorrer em certas ocasiões, dados reais são específicos em seu domínio e podem não representar o desempenho dos métodos em geral. Por exemplo, um algoritmo que lida bem com classificação de texto não necessariamente terá desempenho similar ao classificar imagens. Além disso, dados reais são inerentemente limitados em sua quantidade, pois dependem de fatores externos para serem coletados.

Um agravante adicional percebido é o uso exclusivo de dados de uma mesma fonte através de diversos trabalhos. Se os modelos são testados sempre nos mesmos dados, seu desempenho medido não é indicativo da adequação do modelo em geral, mas sim a esses dados específicos.

Para além dos dados, é frequente a dependência de acurácia como única métrica de desempenho. Essa prática, apesar de comum, pode criar pontos cegos e esconder problemas de classificação. Por exemplo, em casos de classes desbalanceadas, situação em que uma classe possui significativamente mais pontos que outra, a acurácia pode reportar um alto desempenho mesmo que o algoritmo tenha falhado catastroficamente em aprender os dados. Essa situação é discutida de forma mais aprofundada na seção 3.2 e demonstrada no decorrer dos experimentos.

Assim, neste trabalho, busca-se realizar comparações sistemáticas de classificadores binários supervisionados como mecanismo para responder à pergunta: é possível fazer uso de dados sintéticos como estratégia para elucidar a heterogeneidade de tais modelos?

1.1 Principais contribuições

Sumariamente, esta dissertação traz as seguintes contribuições:

- Apresenta a implementação de um gerador de dados artificiais – inspirado por trabalhos anteriores – refinado através da adição de ruído, desbalanceamento de dados, rotações dos pontos e o cálculo automático da separabilidade das classes;
- Realiza uma pequena revisão da literatura com intuito de investigar possíveis vantagens do uso de otimização de parâmetros nos classificadores testados;
- Atesta a similaridade de desempenho, em geral, entre os classificadores nos dados em que foram testados;
- Separa os classificadores em grupos em respeito a seus comportamentos similares em certas ocasiões:
  - No que tange à qualidade dos dados, separou-se em dois grupos os algoritmos conforme sua resistência a ruído e desequilíbrio entre classes: no primeiro grupo, *k-Nearest Neighbors* (kNN), *Random Forest* (RF) e *Stochastic Gradient Boosted Decision Trees* (GBDT), mais resilientes; e, no segundo, SVMs e *Gaussian Naive Bayes* (NB), mais vulneráveis a essas características;
  - Em termos de tempo, separou-se os classificadores em três grupos: algoritmos lentos (RF e GBDT), meio-termo (SVMs), e rápidos (kNN e NB);
- Ressalta a necessidade de, ao realizar otimização de dados, fazê-la de maneira informada, levando em consideração os dados a serem classificados, para não obter desempenhos piores do que os classificadores teriam com parâmetros padrão;
- Avalia a adequação de diversas métricas de desempenho, apontando considerações a serem feitas durante sua escolha que podem ser relevantes em trabalhos futuros de classificação e aprendizado de máquina em geral;
- Finalmente, recomenda o uso do kNN como classificador binário supervisionado, caso não haja fatores adicionais a serem considerados, devido a seu bom desempenho, resistência a dados falhos e curto tempo de execução.

1.2 Estrutura do texto

O restante deste documento está organizado como descrito a seguir. O Capítulo 2 indica trabalhos anteriores que lidam com tópicos relacionados ao escopo do trabalho e que serviram como influência para refinar o objetivo desta dissertação.

Os capítulos seguintes lidam com os experimentos do trabalho. A Figura 1.1 contém um fluxograma descrevendo, superficialmente, os focos e diferenças entre os capítulos.

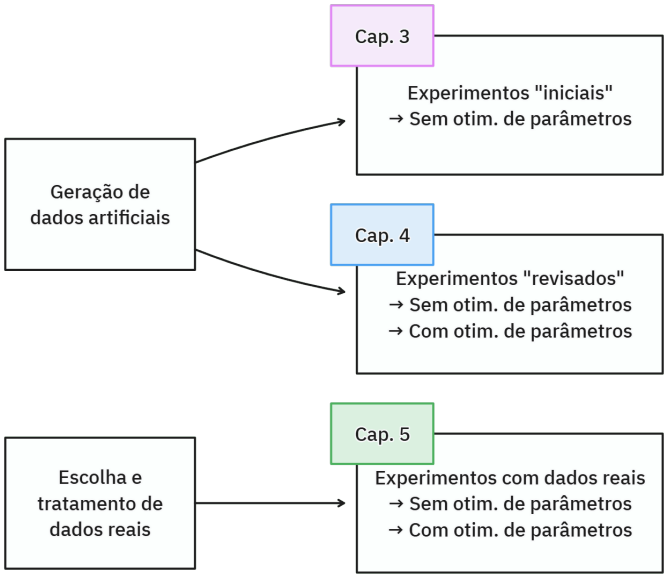


Figura 1.1: Fluxograma descrevendo a estrutura do trabalho através dos diversos experimentos realizados.

Inicialmente, foram realizados experimentos com dados artificiais, de forma a validar a ideia da pesquisa. Estes foram executados sem a otimização de parâmetros, e com um conjunto inicial de seis métricas de desempenho. Tais experimentos são explorados no Capítulo 3.

Os experimentos foram, então, modificados, após a percepção de pontos insatisfatórios nos testes iniciais. Ainda lidando com dados artificiais, há a introdução de otimização de parâmetros e a mudança de algumas das métricas de desempenho. O Capítulo 4 detalha essas diferenças e os experimentos executados.

O Capítulo 5, por sua vez, investiga sucintamente a reação dos classificadores a uma pequena gama de dados reais de fontes distintas, com o intuito de diversificar as situações às quais os algoritmos são submetidos.

Finalmente, no Capítulo 6, há uma discussão dos resultados de todos os experimentos realizados anteriormente. A dissertação é concluída no Capítulo 7, recapitulando o trabalho como um todo e apresentando oportunidades em aberto para trabalhos futuros.

Capítulo 2

Trabalhos relacionados

Comparar o desempenho de classificadores é uma iniciativa que já foi realizada em mútliplas ocasiões anteriormente (KING *et al.*, 1995). Habitualmente, trabalhos que introduzem novos algoritmos realizam experimentos comparativos com classificadores já estabelecidos, justificando a necessidade da introdução de um classificador inédito – ou, pelo menos, seu igual valor em relação aos existentes (JAPKOWICZ e SHAH, 2011).

Contudo, há um viés inerente a tais experimentos no que tange às intenções dos autores: se seus resultados mostram pouca ou nenhuma vantagem sobre algoritmos pré-existentes, então há pouco sentido em publicar um novo classificador. Em sua extensa crítica a trabalhos comparativos anteriores, KING *et al.* (1995) resumem o sentimento eloquentemente como "um cínico diria que a única conclusão geral que pode ser feita de tais estudos é que algoritmos em que os autores possuem o maior interesse tendem a obter os melhores resultados".

Verifica-se que há um estímulo para que o pesquisador publique somente ocasiões em que seu algoritmo possui bom desempenho, e omita situações desfavoráveis. JAPKOWICZ e SHAH (2011) traçam um paralelo com estudos farmacêuticos, evidenciando a separação entre pesquisa e teste nesta outra área. Os autores argumentam que pesquisadores envolvidos na produção de novos medicamentos normalmente não são também responsáveis pelos testes formais realizados para comprovar sua eficácia e aprovar seu uso: "eles conduzem testes informais prototípicos para guiar sua pesquisa, mas deixam a testagem formal para outros pesquisadores melhor treinados em métodos estatísticos" (JAPKOWICZ e SHAH, 2011).

Em contraste, os autores apontam que pesquisadores de aprendizado de máquina possuem maior liberdade para customizar e iterar seus algoritmos, e isso os leva a subestimar testes estatísticos formais. Assim, argumentam, tais testes serão fundamentalmente enviesados.

Adicionalmente, JAPKOWICZ e SHAH (2011) avaliam exaustivamente testes comparativos anteriores e criticam duas características frequentes em especial: a

utilização de uma única métrica de desempenho, comumente a acurácia, e a dependência de *datasets* exclusivamente do repositório da UCI[1] para todos os testes.

Os autores argumentam que a forma de medir o desempenho de um algoritmo é tão importante quanto a escolha dele próprio. Ademais, a exclusiva utilização de *datasets* pré-prontos da UCI permite a criação de pontos cegos ou até *overfitting*.

Neste quesito, KING *et al.* (1995) fazem críticas similares anos antes. Os autores citam, em trabalhos anteriores, a baixa diversidade de algoritmos sendo comparados; o diminuto tamanho ou quantidade de conjuntos de dados distintos utilizados para a comparação; a falta de transparência em qual implementação de cada algoritmo está sendo utilizada; e a prática, às vezes não anunciada, de otimização de parâmetros (*parameter tuning* ou *tweaking*). Otimizar parâmetros para alguns algoritmos, mas não para outros, pode criar vantagens sintéticas que, os autores argumentam, afetam significativamente os resultados.

FERNÁNDEZ-DELGADO *et al.* (2014) analisam uma quantidade colossal de classificadores – 179, segundo reportam – implementados em C/C++, Matlab, R e Weka, e treinados na íntegra dos *datasets* da UCI (108 dos 112 *datasets* usados, ou 96.43%) somada a 4 outros conjuntos de dados reais. Os autores encontram que algoritmos de RF possuem o melhor desempenho, seguidos daqueles de SVM. Contudo, o trabalho sofre das mesmas falhas descritas anteriormente: o desempenho dos algoritmos é medido exclusivamente por sua acurácia na classificação, e os dados utilizados são quase inteiramente advindos do repositório da UCI.

ZHANG *et al.* (2017), por sua vez, percebem uma falha de inclusão de algoritmos mais recentes ou bem avaliados em pesquisas comparativas anteriores, como *Extreme Learning Machine* (ELM), *Deep Learning* (DL) ou GBDT. Assim, eles se propõem a conferir se RF e SVM realmente possuem melhor *performance*, ou se algum classificador mais recente é mais recomendado. Os autores executam experimentos com 71 *datasets* da UCI, KEEL e LibSVM, fazendo uso de validação cruzada de 10 *folds* e separando cada conjunto de dados em 80% para treino, 10% para otimização de parâmetros, e 10% para teste.

Os autores reportam que GBDT, RF e SVM performam melhor quando medidos por acurácia e *Area Under ROC Curve* (AUC). Também é salientado que tanto acurácia quanto AUC costumam concordar com a ordem (ou *ranking*), de melhor a pior, dos algoritmos testados. Entretanto, os autores encerram o artigo enfatizando que mais métricas de desempenho devem ser consideradas para que seja possível generalizar os resultados para problemas arbitrários.

OH (2011), em sua proposta de introdução de um novo algoritmo para calcular a sobreposição de classes em um conjunto de dados, realiza experimentos para estimar

[1]UCI Machine Learning Repository. *University of California, Irvine*. Disponível em: `https://archive.ics.uci.edu/`. Acesso em 22 jan.2024.

a correlação entre o desempenho de classificadores e a sobreposição dos dados classificados. Não surpreendentemente, o autor encontra uma relação inversa: maiores acurácias tendem a estar associadas a menores sobreposições de dados, e vice-versa.

Na Tabela 2.1, diversos trabalhos comparativos anteriores são listados, salientando classificadores, métricas de desempenho e conjuntos de dados utilizados para as comparações; as informações referentes ao trabalho atual também estão presentes. O significado das abreviações está explicitado na Lista de Abreviaturas.

Tabela 2.1: Comparações anteriores de classificadores

Artigo	**Classificadores**	**Métricas**	**Observações**
AMANCIO *et al.* (2014)	Bayesian Network, C4.5, CART, kNN, LogR, Multilayer Perceptron, NB, RF, SVM	Acurácia	Dados gerados artificialmente pelos autores; uso exclusivo de acurácia como métrica de desempenho.
FERNÁNDEZ-DELGADO *et al.* (2014)	179 classificadores	Acurácia	108 *datasets* do repositório UCI; 4 *datasets* do mundo real (histologia).
OH (2011)	kNN, NB, Neural Network, SVM	Acurácia	10 *datasets* do repositório UCI; uso exclusivo de acurácia como métrica de desempenho.
LIM *et al.* (2000)	22 algoritmos de árvore de decisão; 9 estatísticos; 2 de redes neurais	Acurácia Tempo	16 *datasets* do repositório UCI, com e sem ruído; uso quase exclusivo de acurácia.
SINGH *et al.* (2017)	kNN, NB, RF	Acurácia Tempo	8 *datasets* do repositório UCI, com e sem ruído; uso quase exclusivo de acurácia.
ZHANG *et al.* (2017)	AB, C4.5, DL, ELM, GBDT, kNN, LogR, NB, RF, SRC, SVM	Acurácia AUC Tempo	71 *datasets* dos repositórios UCI, KEEL e LibSVM, com e sem ruído.
Este trabalho	GBDT, kNN, NB, RF, SVM	Acurácia AUC F-score LogLoss MSE RMSE Tempo	Dados gerados artificialmente pelos autores, com e sem ruído, com e sem desequilíbrio de dados. 3 *datasets* dos repositórios UCI, Kaggle e Data.world.

Capítulo 3

Experimentos iniciais

Os experimentos deste trabalho, como um todo, têm como objetivo aferir a adequação de algoritmos de classificação. Nos experimentos iniciais, estes classificadores foram utilizados *out-of-the-box*, isto é, com seus parâmetros padrão de implementação, sem passar pelo processo de *parameter tuning* (ou otimização de parâmetros). Esta expressão se refere ao ajuste manual ou automático de parâmetros para otimizar os algoritmos a um conjunto de dados.

O pensamento por trás dessa decisão foi simples: a introdução de otimização de parâmetros poderia trazer consigo o enviesamento dos resultados. Alguns algoritmos, como SVM, possuem muitos parâmetros, e a inclusão de um parâmetro em detrimento de outro na otimização poderia se tornar fonte de vieses despercebidos.

Para realizar os experimentos, é preciso fazer três escolhas: quais classificadores serão testados, como aferir o desempenho de cada um, e quais dados eles deverão classificar. A escolha de classificadores testados é descrita na seção 3.1. A seção seguinte, 3.2, detalha a escolha de métricas de desempenho.

Finalmente, em termos de dados usados para testar os classificadores, decidiu-se em prol de dados artificiais, como tentativa de prevenir enviesamento e *overfitting* para conjuntos de dados reais específicos. Como JAPKOWICZ e SHAH (2011) explicitam em sua crítica, a dependência de *datasets* pré-prontos em trabalhos anteriores pode causar enviesamento e trazer pontos cegos. Desse modo, o uso de dados sintéticos foi considerado mais adequado para os primeiros experimentos.

O processo para os experimentos foi o seguinte: primeiro, há a geração de dados artificiais. A Figura 3.1 mostra um fluxograma do processo de geração de dados, que é descrito em mais detalhes na seção 3.3. A seção 3.4, por sua vez, descreve o cálculo de sobreposição de dados realizado após a geração dos mesmos. Então, os algoritmos são treinados sobre os dados – etapa descrita na seção 3.5 – e, finalmente, os resultados são coletados e analisados na seção 3.6.

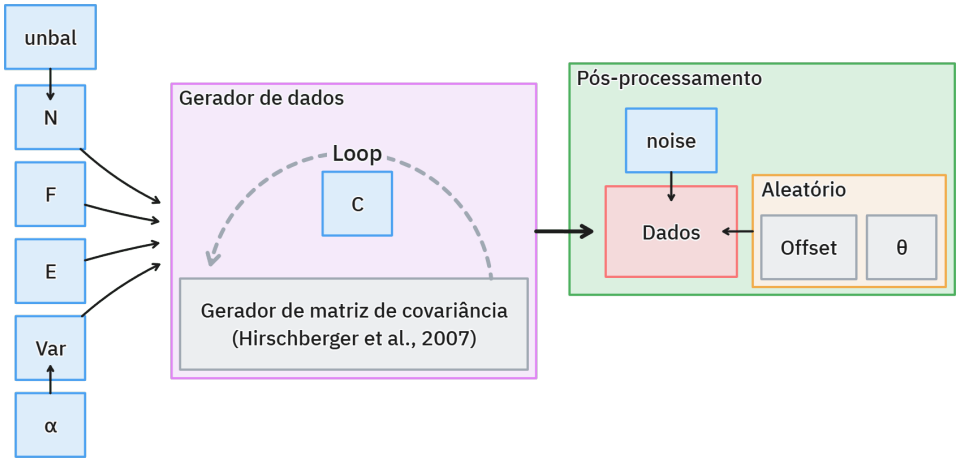


Figura 3.1: Fluxograma descrevendo o processo de geração de dados artificiais. Caixas em azul indicam parâmetros de entrada do sistema; as caixas roxa e verde indicam as duas principais etapas do processo.

3.1 Classificadores supervisionados

No intuito de criar uma avaliação academicamente interessante, foram escolhidos sete classificadores bem avaliados por trabalhos anteriores, distribuídos entre cinco categorias de algoritmo distintas. A Tabela 3.1 lista os métodos escolhidos e os trabalhos considerados para selecioná-los.

As implementações dos algoritmos usadas são da biblioteca de Python 3, scikit-learn.[1] A biblioteca é amplamente utilizada na literatura, e é considerada "o pacote mais compreensivo de aprendizado de máquina com código aberto em Python" (HAO e HO, 2019). Mais detalhes do ambiente de desenvolvimento são descritos na seção 3.5.1.

As minuciosidades por trás do funcionamento de cada algoritmo escolhido já foram detalhadas *ad nauseam* em trabalhos anteriores (ABU-MOSTAFA *et al.*, 2012 (e-Chapters 6 e 8);[2] CHEN *et al.*, 2008; DOMINGOS e PAZZANI, 1997; FAWAGREH *et al.*, 2014), e fogem um pouco ao escopo do trabalho. Aqui, descreve-se resumidamente os modelos selecionados:

SVM O modelo, em sua essência, enxerga o problema de classificação como a otimização de uma função que tenta desenhar, em múltiplas dimensões, para separar as classes (ABU-MOSTAFA *et al.*, 2012: e-Chap. 8). O SVM base

[1] Disponível em: `https://scikit-learn.org/`. Acesso em 2 jul.2024. SVM via 'sklearn.svm.SVC'. Documentação sobre o *kernel* pré-computado disponível em: `https://scikit-learn.org/stable/modules/svm.html#using-the-gram-matrix`. GBDT via 'sklearn.ensemble.GradientBoostingClassifier'. RF via 'sklearn.ensemble.RandomForestClassifier'. kNN via 'sklearn.neighbors.KNeighborsClassifier'. NB via 'sklearn.naive_bayes.GaussianNB'.

[2] Disponível em: `https://amlbook.com/eChapters.html`. Acesso em 26 nov.2024.

procura uma separação linear entre os pontos utilizando um hiperplano; contudo, a introdução de *kernels* transforma o espaço no intuito de remover essa restrição (ABU-MOSTAFA *et al.*, 2012: e-Chap. 8).

GBDT *Boosting* é o processo em que cria-se uma sequência de classificadores fracos, cada um treinado para classificar pontos incorretamente classificados pelas instâncias anteriores, na esperança de terminar com um modelo forte (FAWAGREH *et al.*, 2014). Em *Gradient Boosted Decision Trees*, usa-se árvores de regressão ajustadas ao gradiente descendente da função de erro até o modelo convergir. O aspecto estocástico entra quando cada árvore individual treina com uma amostragem aleatória dos dados de treino (CHEN *et al.*, 2008).

Random Forest O algoritmo de *Random Forest* é um exemplo de *ensemble learning* – onde múltiplos classificadores são usados para resolver um mesmo problema – fazendo uso de *bootstrap aggregating* (ou *bagging*) – onde cada classificador recebe uma seleção aleatória dos dados, e possui igual poder de voto para sua classificação (FAWAGREH *et al.*, 2014). Os classificadores, em si, são árvores de decisão.

kNN A lógica por trás do kNN é extremamente simples: para classificar um ponto x, encontre os K pontos mais próximos a x no conjunto de treino, e veja qual a classe mais frequente dentre eles; essa será a classe de x (ABU-MOSTAFA *et al.*, 2012: e-Chap. 6).

Naive Bayes Similarlmente ao kNN, o modelo de *Naive Bayes* é muito simples. Adaptando a descrição de DOMINGOS e PAZZANI (1997), para a i-ésima classe C_i, o algoritmo calcula uma função discriminante f_i, definida a seguir, que determina a probabilidade estimada de um ponto X estar na classe (onde A_j representa o j-ésimo atributo dos dados, e X_j seu valor no ponto a ser classificado):

$$f_i(X) = P(C_i) \prod_j P(A_j = X_j | C_i)$$

A grande desvantagem do algoritmo é a presunção da independência dos atributos dos dados; contudo, NB ainda consegue ser um algoritmo ótimo em situações em que os atributos não são independentes (DOMINGOS e PAZZANI, 1997).

Também é relevante explicitar os parâmetros usados em cada modelo. Como descrito anteriormente, os parâmetros padrão da implementação foram mantidos, e estes estão descritos na Tabela 3.3.[3]

[3] O algoritmo GBDT requer, pela implementação, um valor para o parâmetro subsample < 1 para que tenha comportamento estocástico. Assim, este foi selecionado como 0.5.

Tabela 3.1: Algoritmos de classificação escolhidos

#	**Classificador**	**Família**	**Referências**
1	Support Vector Machines (SVM), via Sequential Minimal Optimization (implementado na LibSVM)	Function	*(dadas individualmente abaixo)*
1.1	SVM (*kernel* pré-computado)	Function	ZHANG *et al.* (2017)
1.2	SVM (*kernel* gaussiano ou RBF)	Function	FERNÁNDEZ-DELGADO *et al.* (2014); SYARIF *et al.* (2016)
1.3	SVM (*kernel* polinomial)	Function	FERNÁNDEZ-DELGADO *et al.* (2014)
2	Stochastic Gradient Boosted Decision Trees (GBDT)	Boosting / Decision Tree	ZHANG *et al.* (2017); XIA *et al.* (2017)
3	Random Forest (RF)	Decision Tree	FERNÁNDEZ-DELGADO *et al.* (2014); ZHANG *et al.* (2017)
4	K-Nearest Neighbors (kNN)	Nearest Neighbor	AMANCIO *et al.* (2014)
5	Gaussian Naïve Bayes (NB)	Bayesian	AMANCIO *et al.* (2014)

Tabela 3.3: Parâmetros padrão da biblioteca scikit-learn

Classificador	**Parâmetros**
SVM (*kernel* pré-computado)	$C = 1$; gamma $= \frac{1}{F \times Var[X]}$
SVM (*kernel* RBF)	$C = 1$; gamma $= \frac{1}{F \times Var[X]}$
SVM (*kernel* polinomial)	$C = 1$; gamma $= \frac{1}{F \times Var[X]}$; degree $= 3$
GBDT	learning_rate $= 0.1$; n_estimators $= 100$; max_depth $= 3$; max_features $= N$
RF	n_estimators $= 100$
kNN	n_neighbors $= 5$; weights $=$ uniform; metric $=$ minkowski
NB	var_smoothing $= 10^{-9}$

3.2 Métricas de desempenho

Existem diversas formas de medir o desempenho de um classificador. Comumente, usa-se a acurácia que, em termos simples, refere-se à porcentagem de acertos em relação ao total de pontos classificados. Assim, um classificador que corretamente classifica 7 pontos em um conjunto de 10 alcança 70% de acurácia nessa situação.

Ainda que possa ser um indicador útil, o uso exclusivo de acurácia para a avaliação de algoritmos pode ser problemático a depender dos dados sendo classificados. Este é o caso de *datasets* desequilibrados, em que o número de pontos através das diferentes classes não é igual.

HE e GARCIA (2009) exemplificam a situação com um conjunto de dados de mamografias, em que $10,923$ instâncias estão na classe negativa (sem câncer), e apenas 260 na classe positiva (com câncer). Um algoritmo que classifique todo e qualquer dado entregue a ele como "negativo" teria, neste exemplo, uma acurácia de 97.68%, errando apenas 2.32% das vezes. Ainda assim, ele estaria errado para 100% dos casos positivos e nunca acusaria a possibilidade de câncer, o que poderia ter consequências gravíssimas. HE e GARCIA (2009) resumem sua análise argumentando que, em casos de dados desbalanceados, "a prática convencional de usar um único critério de avaliação", comumente a acurácia, "não proporciona as informações adequadas".

Assim, é interessante diversificar as métricas de desempenho dos classificadores, e não restringir-se a somente uma. Para determinar quais métricas são adequadas, pode-se utilizar estudos anteriores que tratam exatamente da comparação de métricas de desempenho, como os de FERRI *et al.* (2009) e LABATUT e CHERIFI (2012).

FERRI *et al.* (2009) comparam 18 métricas de desempenho usando seis algoritmos atuando em *datasets* variados. Este trabalho foi a base da seleção das métricas de LogLoss, AUC e MSE, levando em consideração sua eficácia. Além disso, os autores oferecem alguns *insights* sobre o comportamento das métricas: acurácia e outras métricas ditas qualitativas têm melhor aplicação quando há ruído nos dados, quando comparadas com medidas probabilísticas; contudo, quando usadas com um algoritmo desconfigurado ou *datasets* de treino pequenos, essas mesmas métricas são inadequadas, e a medida de AUC é a melhor escolha.

LABATUT e CHERIFI (2012) estudam por volta de 10 métricas de desempenho e apontam que diversas delas são similares, ou até possuem relação linear entre si, e, portanto, a escolha entre uma ou outra é irrelevante. Assim, argumentam que, na falta de um estudo estatístico aprofundado, a melhor escolha é aquela mais simples, cuja interpretação é a mais clara. Os autores concluem com a defesa do uso de acurácia, para a avaliação de acertos em geral, e F-score, caso seja interessante

analisar uma classe específica.

A Tabela 3.5 lista as métricas escolhidas para os experimentos iniciais e os estudos considerados para selecioná-las.

Tabela 3.5: Métricas de desempenho escolhidas

#	**Método**	**Implementação**	**Referências**
1	Acurácia / Overall Success Rate	Via scikit-learn: 'sklearn.metrics.accuracy_score'	KING *et al.* (1995); FERRI *et al.* (2009); LABATUT e CHERIFI (2012)
2	F-score / F-measure	Via scikit-learn: 'sklearn.metrics.f1_score'	LABATUT e CHERIFI (2012)
3	LogLoss	Via scikit-learn: 'sklearn.metrics.log_loss'	FERRI *et al.* (2009)
4	Area Under ROC Curve (AUC)	Via scikit-learn: 'sklearn.metrics.roc_auc_score'	FERRI *et al.* (2009)
5	Mean Squared Error (MSE)	Via scikit-learn: 'sklearn.metrics.mean_squared_error'	FERRI *et al.* (2009)
6	Tempo de treino	Via Python 3.7+: 'time.monotonic_ns'	KING *et al.* (1995)

3.3 Geração de dados

3.3.1 Requisitos básicos

Nos experimentos iniciais, levando em consideração as críticas feitas por JAPKOWICZ e SHAH (2011) a trabalhos anteriores, o uso de conjuntos de dados pré-prontos foi evitado – em especial os mais utilizados, oferecidos pelo repositório da UCI. Assim, faz-se necessário um gerador de dados artificiais que permita algum grau de liberdade para a geração de dados interessantes. Em sua forma mais simples, o gerador deve, pelo menos, permitir a escolha de:[4]

- Quantidade total de pontos a serem gerados ($N \in \mathbb{N}^*$);
- Quantidade de dimensões de cada ponto ($F \in \mathbb{N}^*$);
- Número de classes em que os pontos serão colocados ($C \in \mathbb{N}^*$).

Dentre a gama de opções para a geração de dados artificiais, uma escolha influente a esta pesquisa foi a de AMANCIO *et al.* (2014). A partir de um método de

[4]Onde $\mathbb{N}^*$ indica o conjunto de números naturais sem zero; i.e. $\{1, 2, 3, ...\}$.

geração de matrizes de covariância aleatórias anterior, proposto por HIRSCHBERGER *et al.* (2007), os autores o adaptaram para que usasse as matrizes geradas para criar dados sintéticos de forma eficiente.

HIRSCHBERGER *et al.* (2007) disponibilizam o seguinte método, que permite a geração de uma matriz de covariância aleatória com quatro parâmetros controlados: N, M, e e v. Aqui, N e M são o número de pontos e de dimensões, respectivamente; e é o valor esperado para elementos fora da diagonal principal da matriz final, e v sua variância. O método, extraído em partes selecionadas do trabalho de HIRSCHBERGER *et al.* (2007), é o seguinte:

$$\hat{e} = \sqrt{\frac{e}{N}} \tag{6}$$

$$\hat{v} = -\hat{e}^2 + \sqrt{\hat{e}^4 + \frac{v}{N}} \tag{7}$$

$$q_{ij} \sim \text{Normal} \qquad i \in [1, M], j \in [1, N] \tag{§6.1a}$$

$$f_{ij} = \hat{e} + \sqrt{\hat{v}} q_{ij} \qquad i \in [1, M], j \in [1, N] \tag{§6.1b}$$

$$\Sigma = FF^\top \tag{§6.1c}$$

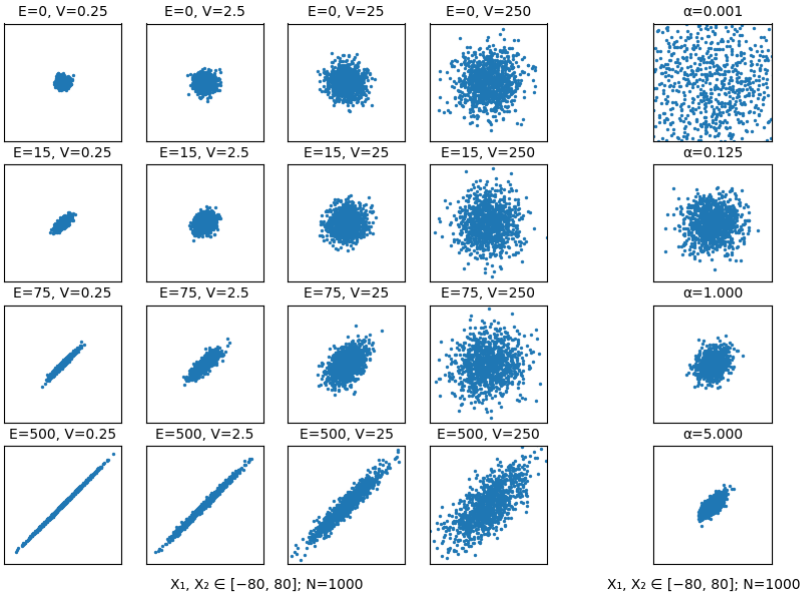
A implementação deste trabalho, em Python, do gerador de dados baseado no método de HIRSCHBERGER *et al.* (2007) está disponível no Apêndice A. A distribuição normal, por não ter seus parâmetros especificados no artigo, foi tratada como Normal$(0, 1)$.

O impacto da alteração dos parâmetros de valor esperado e variância é evidenciado na Figura 3.2(a). Nos experimentos realizados para este trabalho, ambos os parâmetros são escolhidos de forma uniformemente aleatória em um intervalo.

Para cada classe que deve ser gerada, o algoritmo cria uma nova matriz de covariância. Assim, o gerador de dados possui, até então, os seguintes cinco parâmetros: número de pontos N (por padrão, 500), número de dimensões M (p.p. 2), número de classes C (p.p. 2) e dois intervalos de valores possíveis, para calcular valor esperado e variância (p.p. $[0, 50]$ e $[0.25, 25]$, respectivamente).

Um último parâmetro adicional, inspirado pelo trabalho de AMANCIO *et al.* (2014), é α, que pode ser interpretado como a separação interclasse esperada. Ele é simplesmente um escalonamento do valor obtido para a variância: após escolhido um número uniformemente aleatório, este é multiplicado por $\frac{1}{\alpha}$ e se torna a variância escolhida.

Quanto maior α, menor a variância resultante e, consequentemente, mais restrito o *cluster* da classe. Isso, por sua vez, significa que mais provavelmente as classes



(a) Efeito da alteração dos parâmetros de valor esperado e variância (aqui, E e V), variando de 0 a 500 e 0.25 a 250, respectivamente.

(b) Efeito da alteração do parâmetro de separação esperada α, variando de 0.001 a 5.

Figura 3.2: Variação de parâmetros do gerador de dados.

serão facilmente separáveis – e, a partir dessa inferência, vem o nome de "separação esperada". A Figura 3.2(b) evidencia o impacto da variação do parâmetro.

Já possuindo os pontos, é possível transformá-los conforme necessário; as transformações realizadas são discutidas na seção seguinte.

3.3.2 Transformações adicionais

Além dos requisitos básicos, é importante que o método de geração de dados também permita o controle de algumas transformações, para melhor distinguir diferentes conjuntos de dados. As transformações implementadas com esta finalidade foram as seguintes:

- Possibilitar dados desbalanceados (e.g. 75% classe A, 25% classe B);
- Translação e rotação dos *clusters* de pontos;
- Diferentes níveis de ruído.

De início, pode ser especificado ao gerador uma proporção de desequilíbrio de dados em $[0, 1)$, via o parâmetro *unbal*, para que as classes possuam número diferente de pontos. Nos experimentos realizados, tendo em vista o número de pontos N

utilizado, o limite superior de 0.7 para o parâmetro se mostrou adequado para os testes: alto o suficiente para evidenciar dificuldades de classificadores com *datasets* desequilibrados, mas não alto demais ao ponto que os problemas se tornassem a falta de dados.

Na maneira em que os *datasets* são construídos, todos os *clusters* de pontos são gerados com centro na origem. Assim, uma das primeiras preocupações após a geração dos pontos é distanciar os *clusters*. Para tal, para cada classe é calculado um valor uniformemente aleatório em $[-20, 20]$ para cada dimensão, tendo em vista a escala dos agrupamentos gerados, e este é somado à posição de cada ponto.

O passo seguinte é a rotação aleatória dos *clusters*: esta é de magnitude uniformemente aleatória em $[0°, 360°)$, e tem como plano de rotação duas das dimensões dos pontos originais, escolhidas também aleatoriamente.

Em seguida, quando requisitado via a presença do parâmetro *noise*, valores aleatórios, seguindo uma distribuição Normal$(0, \text{noise})$, são somados a cada coordenada de cada ponto. Este passo encerra a geração de dados que, então, possui oito parâmetros: número de pontos N; número de dimensões M; número de classes C; dois intervalos de valores possíveis, para calcular valor esperado e variância; separação esperada α; desequilíbrio *unbal*; e ruído *noise*.

3.4 Análise preliminar dos dados gerados

Visando facilitar a análise posterior de desempenho dos algoritmos, é possível obter algumas métricas adicionais sobre os dados gerados. Para esse fim, é calculada a sobreposição entre as diferentes classes – de certa forma, uma estimativa de quão linearmente (in)separáveis são as classes.

Existem várias formas de aferir a sobreposição de dois conjuntos de dados. Dentre as opções, a escolhida, por sua versatilidade e versão interclasse, foi a medida de R-value de OH (2011).

A medida de R-value é, em sua concepção, simples. Para cada ponto do *dataset*, ela faz a aferição dos vizinhos próximos a ele. Se forem majoritariamente pontos de uma mesma classe, o autor argumenta, há pouca sobreposição no local. Caso contrário, se os pontos vizinhos não forem homogêneos, isso é indicativo de sobreposição. Realizando esse teste para todos os pontos, é possível calcular a proporção de sobreposição para a classe como um todo.

Ela é formalmente definida da seguinte forma – apresentada, aqui, levemente modificada para maior legibilidade:

$$R(C_i, C_j) = \frac{r(C_i, C_j) + r(C_j, C_i)}{|C_i| + |C_j|} \tag{14}$$

$$r(C_a, C_b) = \sum_{m=1}^{|C_a|} \lambda(|\text{kNN}(C_{a,m}, C_b)| - \theta) \tag{15}$$

Onde $\lambda(x) = \begin{cases} 1 : \text{se } \ x > 0; \\ 0 : \text{c.c.} \end{cases}$

Onde $\text{kNN}(\text{ponto}, \text{classe})$ é o conjunto dos K vizinhos mais próximos de *ponto*, subtraídos vizinhos que não pertençam à *classe*.

O algoritmo recebe, além dos pontos devidamente separados em suas classes, dois parâmetros: K e θ. K é o parâmetro que controla quantos vizinhos mais próximos de cada ponto serão avaliados para o cálculo e, θ, por sua vez, define a quantidade a partir da qual vizinhos de outra classe serão considerados. Isto é, quando $\theta = 2$, um ponto só é contado no somatório se possuir 2 ou mais vizinhos (dentre os K mais próximos) na classe oposta.

Em concordância com os experimentos e práticas reportadas pelo autor no artigo, os valores selecionados para os parâmetros foram $K = 7$ e $\theta = 3$.

Durante testes preliminares, percebeu-se que a medida de R-value somente retornava valores acima de aproximadamente 0.6 para dados idênticos e perfeitamente (ou quase) alinhados. Assim, na implementação, o valor usado para sobreposição é, na verdade, R-value $\times$ 2, limitado superiormente a 1. A implementação deste trabalho, em Python, da função de R-value em si está presente no Apêndice B. Exemplos de R-value aplicados ao gerador de dados artificiais estão representados na Figura 3.3.

Como o algoritmo de R-value contém o que é, essencialmente, um classificador supervisionado de vizinhos mais próximos inteiro, executado para todos os pontos em ambas as classes, a geração de dados foi implementada de forma a permitir que, se desejado, não seja feito o cálculo de sobreposição, visando economizar tempo de processamento quando necessário.

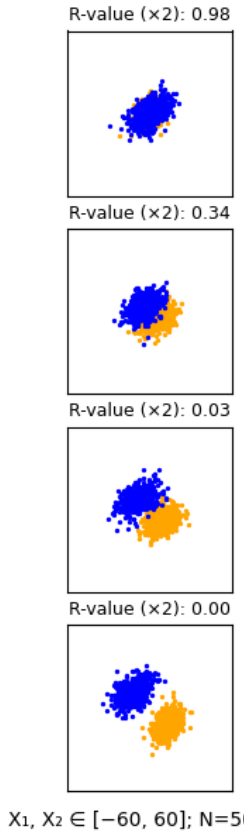


Figura 3.3: Cálculo do R-value para diferentes conjuntos de dados.

3.5 Treino

Após a geração dos dados com os parâmetros especificados, há a etapa de treino. Cada classificador é treinado em 75% dos pontos gerados na etapa anterior e seu tempo de treino é registrado, valor que atua como a primeira métrica de desempenho calculada. Em seguida, os 25% dos pontos restantes são usados para testes de desempenho do classificador já treinado, utilizando as outras cinco métricas escolhidas anteriormente – acurácia, F-score, LogLoss, AUC e MSE.

Os valores dos parâmetros do gerador foram sistematicamente modificados, sendo gradualmente incrementados, para explorar o universo de combinações possíveis de valores. Além disso, cada conjunto de parâmetros foi utilizado em pelo menos 20 execuções de testes, permitindo uma maior exposição às possibilidades da aleatoriedade. O resultado de cada execução foi registrado em um banco de dados SQLite[5] para a permitir uma análise posterior. Ao final dos experimentos, o banco de dados terminou com 185,738 linhas, cada uma contendo informações do desempenho de *um* classificador para *uma* execução de teste usando *um* conjunto de parâmetros do gerador de dados.

[5]Disponível em: `https://sqlite.org/`. Acesso em 5 jul.2024.

3.5.1 Ambiente

Os testes foram realizados no Google Colaboratory,[6] serviço de máquina virtual sob demanda, executando Jupyter Notebook com Python v3.10.12, bibliotecas scikit-learn[7] v1.5.0 e SciPy[8] v1.11.4. A máquina virtual possui dois CPUs Intel® Xeon® @ 2.20 GHz single core com 2 threads, e 12 GB de RAM.

A escolha pelo Google Colaboratory, em vez de execução local, deu-se por diversos motivos. Primeiramente, a dedicação exclusiva dos recursos na máquina virtual, em contraste com a concorrência de diversos processos e programas em um computador pessoal, traz maior estabilidade para os testes, contribuindo, por exemplo, para maior imparcialidade das aferições de tempo de execução. Além disso, há a conveniência inerente de execução em nuvem, pois permite testes ainda que um computador pessoal esteja indisponível – como ocorreu com um dos autores no início do desenvolvimento do projeto, em março de 2023.

3.6 Resultados

Inicialmente, ingenuamente, o foco do trabalho era encontrar diferenças diretas entre o desempenho dos algoritmos. Isto é, determinar se existia um algoritmo "melhor" que todos os outros ou, inversamente, um que deveria ser evitado por apresentar desempenho inferior aos outros.

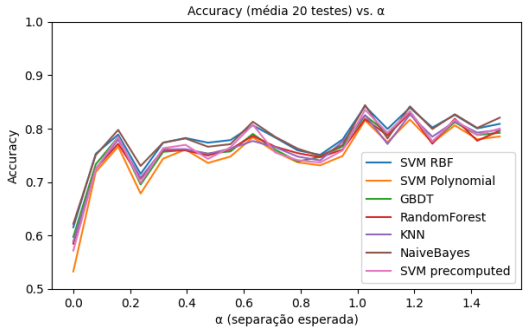
Contudo, com a experimentação foi possível perceber a extrema similaridade de desempenho entre todos os algoritmos testados para os dados sintéticos. A Figura 3.4 sobrepõe o resultado dos sete algoritmos, através das avaliações pelas seis métricas utilizadas, conforme o parâmetro de separação esperada, α, é variado.

De maneira geral, todos os algoritmos são suficientemente adequados para o problema de classificação binária – o que não é surpreendente. Isso é, em parte, explicado pelo viés de sobrevivência: algoritmos com desempenhos significativamente abaixo dos outros não se tornariam populares e, portanto, receberiam menos atenção, menos estudos, menos implementações.

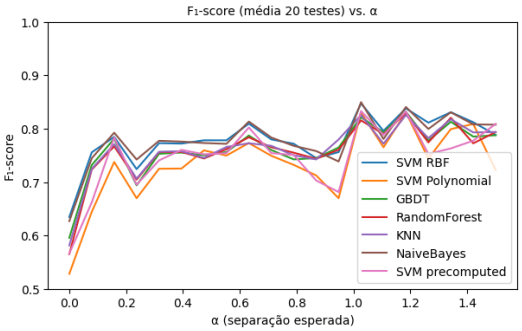
A diferença mais significativa entre os algoritmos é percebida no tempo de treino. Aqui, métodos como RF e GBDT mantêm-se a uma ou duas ordens de grandeza de distância dos seus competidores mais velozes. No caso do SVM com *kernel* pré-computado, o desempenho inicia como o pior de todos mas, no intervalo de $\alpha \in (0, 0.2]$, rapidamente se recupera e se torna razoável.

Entretanto, isso se refere a dados sem ruído; com a gradual inclusão de ruído – vide Figura 3.5 –, há uma inversão na direção da curva de desempenho quantitativo,

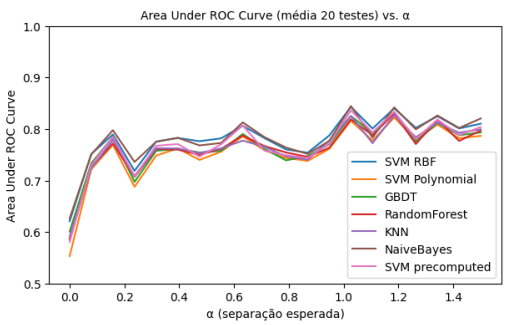
[6]Disponível em: `https://colab.google/`. Acesso em 2 jul.2024.
[7]Disponível em: `https://scikit-learn.org/`. Acesso em 2 jul.2024.
[8]Disponível em: `https://scipy.org/`. Acesso em 8 jul.2024.



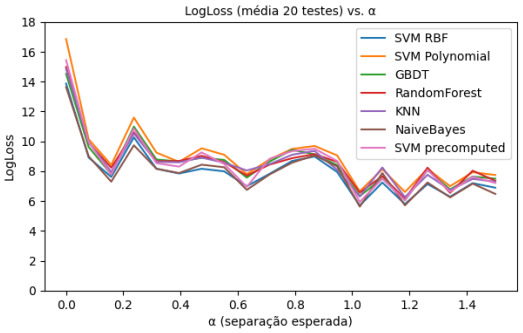
(a) α vs. acurácia.



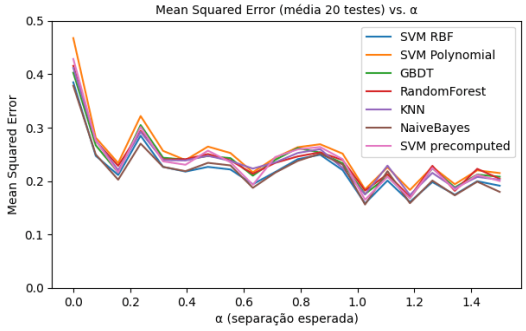
(b) α vs. F-score.



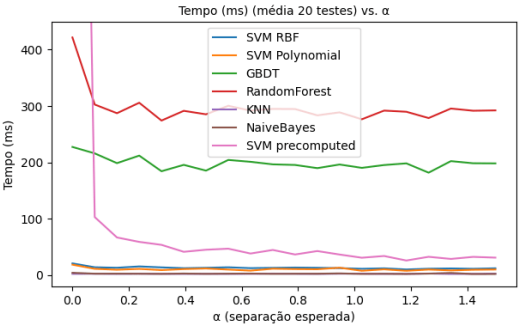
(c) α vs. AUC.



(d) α vs. LogLoss.



(e) α vs. MSE.

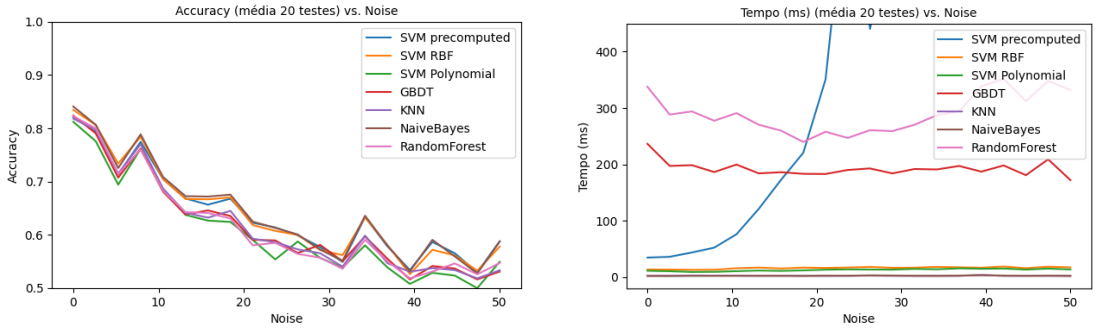


(f) α vs. tempo de treino.

Figura 3.4: Desempenho dos algoritmos conforme α é variado, nas métricas de acurácia, F-score, AUC, LogLoss, MSE e tempo de treino, respectivamente.

como de acurácia. Quando, na variação de α, ela era crescente, agora ela se torna decrescente. Isso traz uma conclusão trivial: o ruído dificulta a classificação correta dos dados.

Além disso, surge uma importante distinção no que tange ao tempo de treino do SVM de *kernel* pré-computado: antes, seu desempenho se iniciava como o pior, e gradualmente se recuperava para a razoabilidade; ordenando por ruído, todavia, o tempo começa na razoabilidade, mas dispara rapidamente para o último lugar.

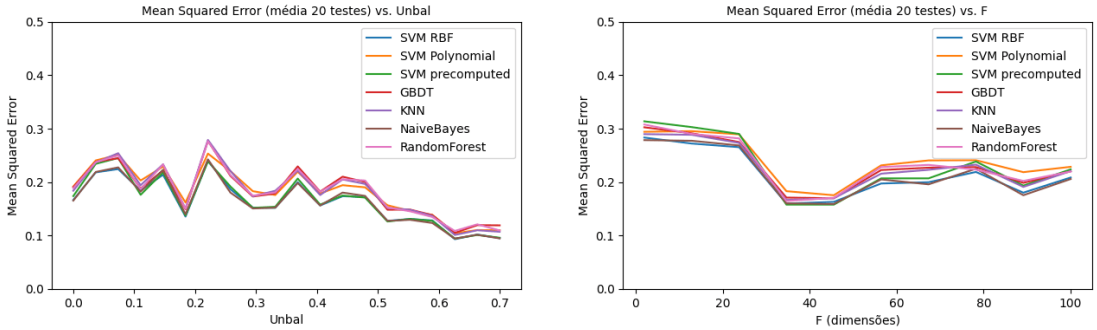


(a) Ruído vs. acurácia.

(b) Ruído vs. tempo de treino.

Figura 3.5: Desempenho dos algoritmos conforme ruído é variado, nas métricas de acurácia e tempo de treino, respectivamente.

A variação de outras variáveis demonstra uma mesma similaridade no desempenho dos algoritmos; as imagens na Figura 3.6 representam o desempenho conforme variam o percentual de desequilíbrio do *dataset* e o número de dimensões, F, respectivamente.



(a) Desequilíbrio vs. MSE.

(b) F vs. MSE.

Figura 3.6: Desempenho dos algoritmos conforme desequilíbrio e número de dimensões são variados, respectivamente, pela métrica de MSE.

3.6.1 Similaridade entre métricas de desempenho

Depender exclusivamente de uma única métrica, como descrito por JAPKOWICZ e SHAH (2011), não é recomendado. Contudo, isso traz à tona o questionamento de

quão similar, entre si, são as diferentes métricas de desempenho.

Para cada uma das seis métricas utilizadas neste capítulo, incluindo tempo, a Figura 3.7 mostra sua relação com a acurácia. Isto é, no eixo horizontal está o desempenho medido pela acurácia e, no vertical, o mesmo desempenho medido pela respectiva métrica.

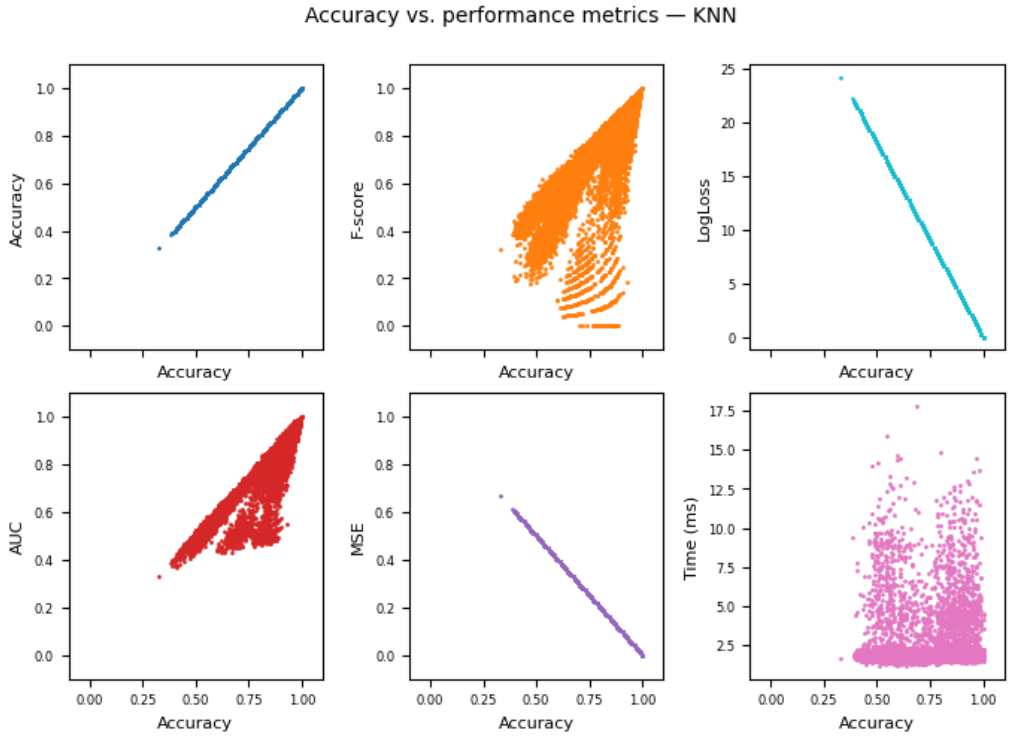


Figura 3.7: Medições de cada métrica de desempenho utilizada vs. acurácia.

Notavelmente, para MSE e LogLoss, a relação é essencialmente linear, ainda que sejam inversamente proporcionais. Pontua-se, no entanto, que essas métricas diferem em seus domínios.

No caso de F-score e AUC, a relação não é tão simples – ainda que, como esperado, uma maior acurácia está em geral associada a um melhor desempenho em F-score e AUC. É interessante notar, contudo, que há situações em que uma acurácia de 90 a 100% está associada a um F-score abaixo de 20%, ou a um AUC próximo a 50%. Isso é posteriormente analisado na seção 6.1.

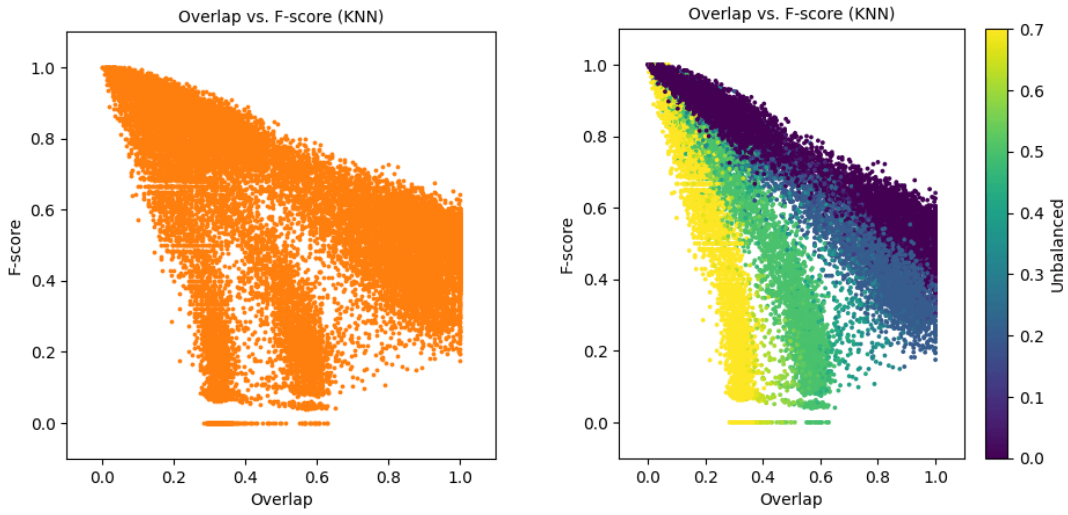
Finalmente, ainda que possa ser esperado, vale notar que não há relação clara entre tempo de treino e acurácia (ou outra métrica). Isto é relevante, pois pode-se acreditar que algoritmos que tiveram "mais tempo" para treino teriam a capacidade de alcançar maiores desempenhos; porém, a princípio, não é o encontrado.

3.6.2 Meta-análise

Visto que não há uma resposta direta para a questão de melhor classificador, a pesquisa foi redirecionada para considerar o impacto de cada parâmetro no desempenho dos classificadores, e na busca por relações que pudessem ser exploradas.

Alguns *insights* interessantes foram descobertos. Por exemplo, há uma relação sofisticada entre o desequilíbrio do conjunto de dados, sua sobreposição e o desempenho dos algoritmos sobre ele. Ambas as imagens na Figura 3.8 indicam o F-score calculado para as classificações do algoritmo de kNN para diversas configurações do gerador de dados, colocado em contraste à sobreposição (calculada *a posteriori*). A Figura 3.8(b) pinta esses dados de diferentes cores de acordo com a variação do desequilíbrio.

Percebe-se na Figura 3.8(a) que, conforme aumenta a sobreposição de dados, o F-score máximo alcançado diminui. Quando em 0% de sobreposição, o F-score está quase inteiramente em 100%; ao alcançar 100% de sobreposição, o F-score encontra-se por volta de 45±15%. Indo além, na Figura 3.8(b) verifica-se que, conforme o desequilíbrio dos dados aumenta – indicado por uma cor mais clara dos pontos –, a sobreposição máxima encontrada diminui. Isto é, é menos provável encontrar grandes sobreposições de dados quando o conjunto de dados é fortemente desequilibrado. Isso faz sentido, pois uma das classes é muito menor e, portanto, há menos pontos sendo sobrepostos. Uma crítica a partir dessa percepção está presente na seção 6.2.

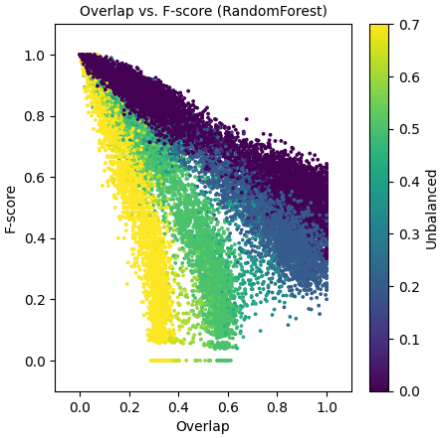


(a) Sobreposição vs. F-score (kNN).

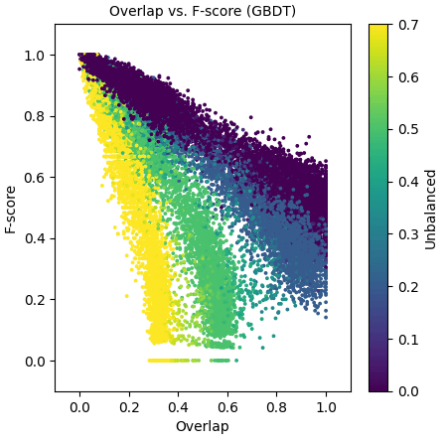
(b) Sobreposição vs. F-score vs. desequilíbrio (kNN).

Figura 3.8: Relação entre a análise de sobreposição vs. métrica de F-score.

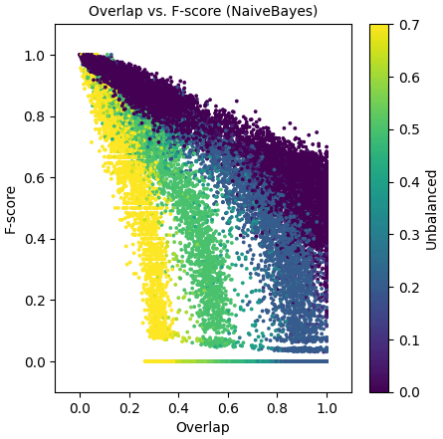
Contudo, isto se refere somente ao desempenho do classificador kNN. Na Figura 3.9, uma mesma visualização é construída para todos os outros classificadores



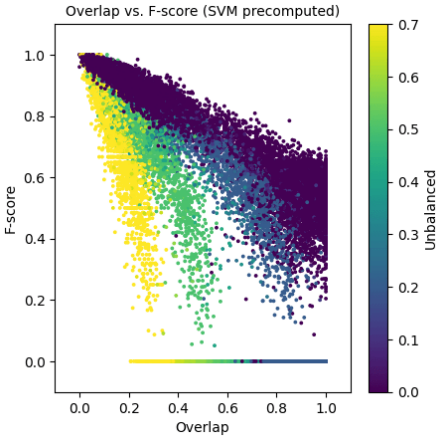
(a) Sobreposição vs. F-score vs. desequilíbrio (RF).



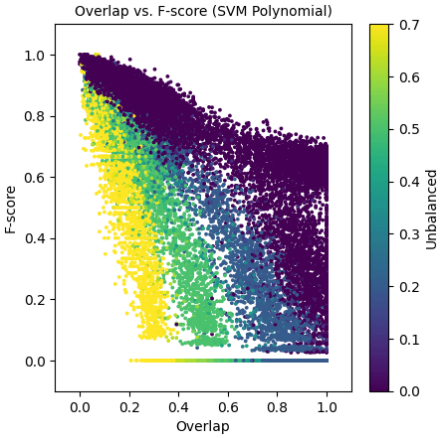
(b) Sobreposição vs. F-score vs. desequilíbrio (GBDT).



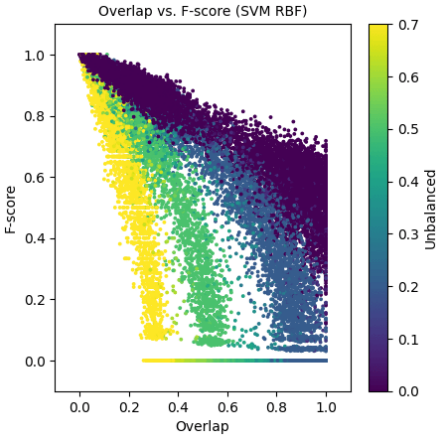
(c) Sobreposição vs. F-score vs. desequilíbrio (NB).



(d) Sobreposição vs. F-score vs. desequilíbrio (SVM pré-comp.).



(e) Sobreposição vs. F-score vs. desequilíbrio (SVM pol.).



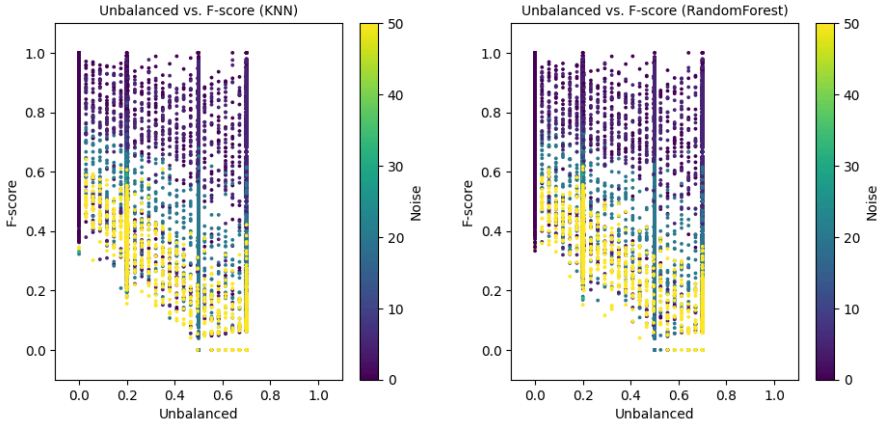
(f) Sobreposição vs. F-score vs. desequilíbrio (SVM RBF).

Figura 3.9: Relação entre sobreposição e F-score avaliando, da esquerda para a direita, de cima para baixo: kNN, RF, GBDT, SVM (*kernel* pré-computado, polinomial, RBF) e NB. Colorido pela porcentagem de desequilíbrio nos dados.

usados. Nota-se que Figura 3.8(b), Figura 3.9(a,b) (kNN, RF e GBDT) são bastante similares. Porém, há algumas distinções perceptíveis no desempenho dos algoritmos de SVM e NB.

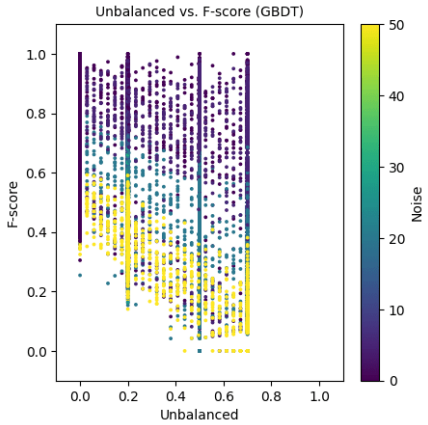
No caso do SVM com *kernel* pré-computado, na Figura 3.9(d), há uma notável ausência de desempenhos com baixo F-score; em grande parte, ou o F-score está acima de 20–30%, ou ele é zero, independentemente de desequilíbrio nos dados.

Para os outros SVMs (polinomial e RBF) e NB, em Figura 3.9(c,e,f), percebe-se uma diferença de comportamento nos casos de alta sobreposição (>80%): o F-score em casos de desequilíbrio médio ($15\% < unbal < 30\%$, cor azul) chega a zero, o que não ocorre para kNN, RF e GBDT. Esta queda de desempenho pode ser indicativa de uma menor resistência ao desequilíbrio de dados, comparativamente.



(a) Desequilíbrio vs. F-score vs. ruído (kNN).

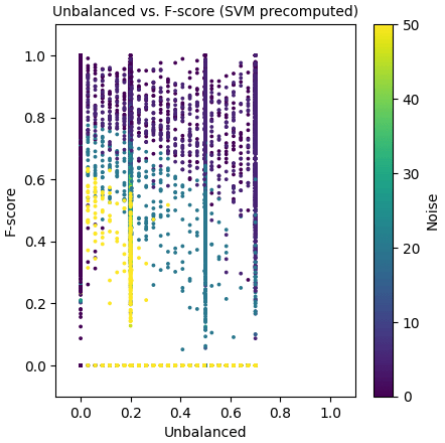
(b) Desequilíbrio vs. F-score vs. ruído (RF).



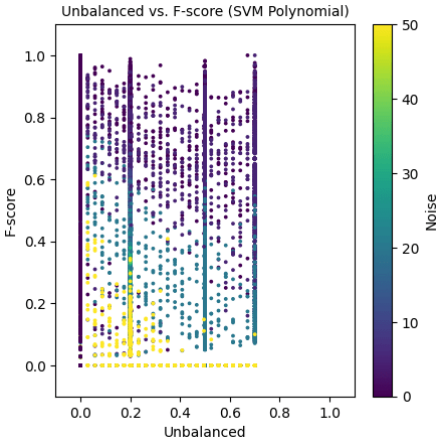
(c) Desequilíbrio vs. F-score vs. ruído (GBDT).

Figura 3.10: Relação entre desequilíbrio e F-score avaliando, da esquerda para a direita, de cima para baixo: kNN, RF e GBDT. Colorido pelo nível de ruído nos dados.

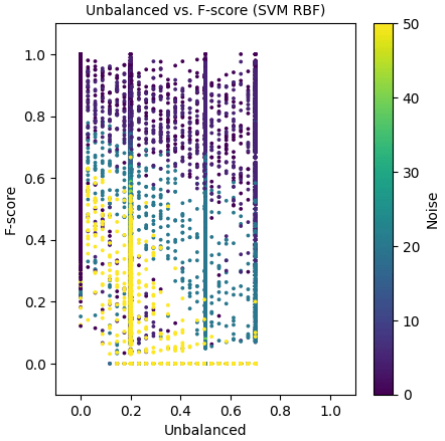
Na Figura 3.10 e na Figura 3.11, encontra-se gráficos de desequilíbrio dos dados



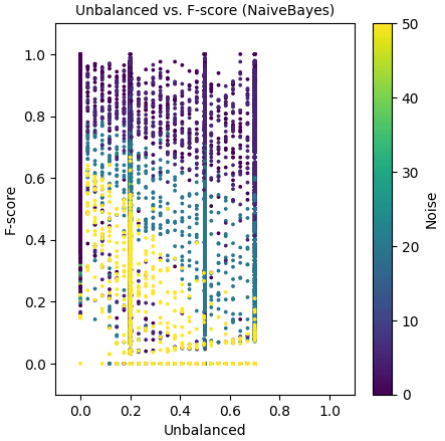
(a) Desequilíbrio vs. F-score vs. ruído (SVM pré-comp.).



(b) Desequilíbrio vs. F-score vs. ruído (SVM pol.).



(c) Desequilíbrio vs. F-score vs. ruído (SVM RBF).



(d) Desequilíbrio vs. F-score vs. ruído (NB).

Figura 3.11: Relação entre desequilíbrio e F-score avaliando, da esquerda para a direita, de cima para baixo: SVM (*kernels* pré-computado, polinomial, RBF) e NB. Colorido pelo nível de ruído nos dados.

vs. F-score, coloridos por nível de ruído. Novamente, kNN, RF e GBDT (Figura 3.10) possuem desempenho muito similar. Comparativamente, percebe-se que, em especial para os algoritmos de SVM (Figura 3.11(a,b,c)), um alto ruído (cor clara) está correlacionado a um desempenho de F-score muito mais baixo, o que não é tão comum nos primeiros três classificadores. Isso pode indicar uma pior habilidade de lidar com ruído alto nos dados, especialmente quando pareado com dados desequilibrados, por parte do SVM, independentemente de *kernel*. O desempenho do classificador de NB (Figura 3.11(d)) é comparável aos SVMs, porém com maior resistência a ruído e desequilíbrio nos dados.

Uma consequência interessante que pode ser percebida é que tanto o desequilíbrio e o ruído do *dataset* quanto a sobreposição das classes podem ser calculados independentemente de qualquer treino e teste dos classificadores. Assim, usando

essas informações, é possível estimar a faixa de desempenho esperada para cada classificador. Inversamente, também é possível calcular algum desses fatores a partir do resultado – por exemplo, estimar o ruído dos dados a partir do desequilíbrio e desempenho do classificador.

Finalmente, realizando uma análise abrangendo todos os experimentos, percebe-se a enorme diferença no tempo de treino de cada classificador. A Figura 3.12 possui um *boxplot* do desempenho, em termos de tempo de treino, de todos os classificadores. Nota-se que o eixo vertical está em escala logarítmica.

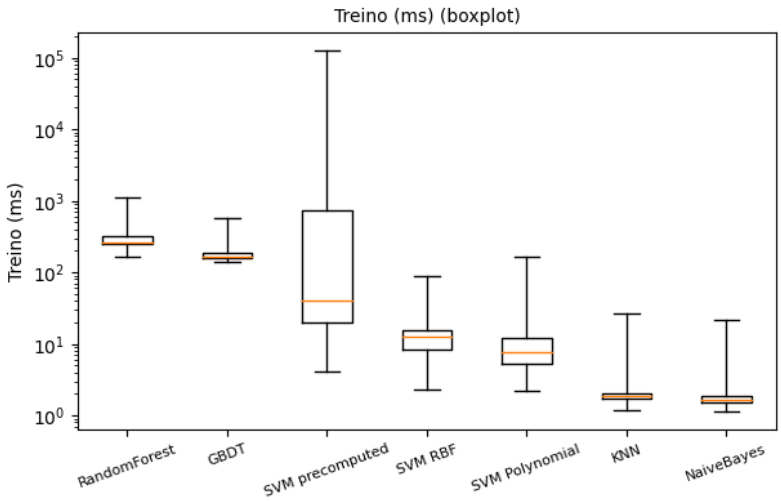


Figura 3.12: *Boxplot* do tempo de treino de cada classificador. O eixo vertical está em escala logarítmica.

Destaca-se, devido à sua alta variância, o SVM com *kernel* pré-computado. Ele alcança tempos de treino significativamente maiores do que todos os outros classificadores, a uma distância de algumas ordens de grandeza. Contudo, em termos de mediana, ele não é o pior desempenho, mas sim o terceiro pior; o último lugar pertence ao RF, seguido de GBDT. No outro lado do espectro, NB é o algoritmo de treino consistentemente mais rápido, seguido de kNN.

No caso do kNN, o bom desempenho é esperado: o kNN não possui etapa de "treino", por dizer, mas, ao invés disso, guarda os pontos de treino para usar em comparações com cada ponto de "teste". Similarmente, NB, em sua etapa de treino, simplesmente computa valores de média e variância para cada classe. Assim, tempo de treino não é uma métrica "justa" – alguns algoritmos são beneficiados, mas não necessariamente são os mais velozes em geral. Essa e outras considerações são discutidas no próximo capítulo.

Seguindo os resultados temporais, pode-se separar os modelos em três categorias, dentre as quais eles são razoavelmente equivalentes: lentos (RF e GBDT), meio-termo (SVMs), e rápidos (kNN e NB). O SVM de *kernel* pré-computado é uma espécie de *outlier* nas categorias, pois possui uma variância muito alta.

Capítulo 4

Experimentos revisados

Os experimentos iniciais, ainda que elucidativos, também evidenciaram alguns problemas. Entre as situações a serem resolvidas, percebeu-se desempenho, de forma geral, muito similar entre os algoritmos; métricas de desempenho linearmente correlacionadas; e tempo de execução pouco representativo. Assim, novos experimentos foram planejados com algumas modificações destinadas a remediar esses problemas. As modificações como um todo são descritas na seção 4.1.

Primeiramente, em uma tentativa de encontrar desempenhos mais díspares entre os classificadores, decidiu-se introduzir a otimização de parâmetros, anteriormente posta de lado sob a justificativa de evitar a introdução de vieses – como a escolha por incluir ou excluir certos parâmetros do processo, de algoritmos que possuem muitos, como o SVM. Contudo, mesmo que haja a possibilidade de enviesamento, a experimentação com otimização de parâmetros ainda é academicamente interessante, e pode servir como base para estudos posteriores mais aprofundados, avaliando a diferença de desempenho a depender de quais parâmetros são otimizados. A seção 4.1.1 detalha as escolhas feitas para essa introdução.

Por sua vez, as métricas de desempenho também foram revisadas, com o intuito de remover a relação linear entre elas, e incluir, além do tempo de execução, o tempo de teste. Esta última situação se mostrou relevante, pois, para algoritmos sem uma fase real de "treino", como kNN e NB, o tempo de treino é evidentemente muito menor; mas é possível que isso seja compensado por um tempo de teste muito maior. Assim, a seção 4.1.2 explica o raciocínio por trás da mudança de métricas, quais foram removidas, adicionadas, e as justificativas por trás de cada decisão.

O gerador de dados sintéticos e a métrica de sobreposição, R-value, mantiveram-se inalterados. A escolha de classificadores, similarmente, não foi modificada – afinal, a avaliação dos algoritmos selecionados continua relevante.

Após as alterações, a seção 4.2 descreve como os experimentos modificados foram realizados. A seção 4.3, por sua vez, apresenta os resultados encontrados nesses novos testes.

4.1 Modificações

4.1.1 Otimização de parâmetros

Uma das maiores mudanças entre os experimentos iniciais e os seguintes é a introdução de otimização de parâmetros, ou *parameter tuning*.[1] Este procedimento, realizado antes da etapa de treino, tenta encontrar valores para os parâmetros de cada modelo que melhorem o desempenho deste para um dado conjunto de dados.

Alguns algoritmos têm grande potencial para serem beneficiados pela prática de otimização de parâmetros; outros, não. Assim, foi preciso determinar para quais classificadores seria interessante realizar o procedimento, considerando que a otimização, por si só, incorre em um tempo de treino adicional não-insignificante.

Contudo, mesmo para classificadores selecionados para otimização, os experimentos foram realizados também com seus parâmetros padrão da implementação, "*out-of-the-box*", como nos experimentos iniciais. Desta forma, é possível avaliar diretamente a diferença de desempenho obtida pela otimização de parâmetros.

Em termos do procedimento escolhido para realizar a otimização de parâmetros, duas opções populares são Grid Search e Random Search – a primeira percorre todos os possíveis valores para cada parâmetro, como definidos pelo implementador, e testa-os um a um; e a segunda realiza um número determinado de testes com valores obtidos a partir de distribuições de probabilidade.

SYARIF *et al.* (2016), estudando *parameter tuning* de SVM, apontam que Grid Search sempre encontra parâmetros quase ótimos para os intervalos usados; contudo, os autores simultaneamente reportam que foram forçados a parar a execução do Grid Search em um *dataset* com muitos pontos, porque sua execução já durava 2 semanas ininterruptas. XIA *et al.* (2017), similarmente, apontam que Grid Search pode ser um processo muito lento e reportam que Random Search alcançou um melhor desempenho em experimentos.

Assim, para o trabalho corrente, a escolha foi em favor do Random Search para todas as otimizações de parâmetros realizadas,[2] com validação cruzada de 5 folds, explorando 25 valores possíveis (nas distribuições especificadas) para cada parâmetro otimizado. Os modelos foram reotimizados a cada nova geração de dados.

Porém, como mencionado anteriormente, nem todo classificador requer otimização. A seguir está um detalhamento aprofundado das considerações feitas para cada classificador e a justificativa por trás da escolha de otimizar ou não seus parâmetros.

[1] Certos autores designam o processo especificamente como "***hyper-parameter*** *tuning*" (ou "*optimization*"), para distinguir "parâmetros" derivados dos dados pelo modelo e "hiper-parâmetros" fixados a priori – vide XIA *et al.* (2017). Entretanto, este trabalho não lida com parâmetros internos aprendidos pelos modelos; assim, aqui, os termos se referem exclusivamente àqueles passados ao algoritmo.

[2] Via 'sklearn.model_selection.RandomizedSearchCV'.

Random Forest PROBST e BOULESTEIX (2018) avaliam os impactos de *parameter tuning* para Random Forest e concluem que a técnica possui impacto "muito menor" do que para outros algoritmos de aprendizado de máquina. Os autores explicitam que o número de árvores deve ser alto pois, quanto mais árvores, melhor o desempenho do modelo; mas, simultaneamente, a melhora é cada vez menor conforme cresce o número de árvores. OSHIRO *et al.* (2012), similarmente, fazem experimentos para avaliar o impacto do número de árvores no desempenho do Random Forest, medido pela métrica AUC. Os autores experimentam com 29 *datasets* de dados médicos reais, com 2 a 16 classes, 3 a 16,000 atributos, e com tamanhos entre 32 e 7200 pontos, incluindo alguns com dados faltantes – então, um espaço de dados bastante abrangente e diverso. Os autores configuram o número de árvores para variar em potências de 2, de 2 a 4096 árvores, e determinam que não há diferença significativa em desempenho a partir de 128; quantidades superiores testadas não trazem benefício que justifique o tempo adicional de processamento. Além disso, os autores percebem que o modelo tende a utilizar todos os atributos presentes no conjunto de dados quando possível, algo que, eles argumentam, pode não ser sempre desejável. Assim, quantidades altas *demais* de árvores podem acabar sendo prejudiciais. Por esses motivos, os autores recomendam um valor fixo entre 64–128. PROBST e BOULESTEIX (2018), também avaliando a importância do número de árvores, declaram que ainda que eles não tenham encontrado malefícios significativos para o aumento irrestrito da quantidade, eles não recomendam tal aumento, pois os ganhos se tornam cada vez menores. O maior ganho de desempenho, eles concluem, está nas primeiras 100–250 árvores. Finalmente, LOUPPE (2015), em uma avaliação de trabalhos anteriores sobre Random Forest, encontra múltiplos estudos afirmando sua robustez para *overfitting*, tornando portanto menos necessário o emprego de alguma técnica de regularização.

Assim, para os experimentos revisados, manteve-se Random Forest sem a otimização de parâmetros; o número de árvores foi fixado em 150.

kNN INYANG *et al.* (2023) realizam experimentos com 33 conjuntos de dados aleatórios da UCI e apontam que um aumento irrestrito do parâmetro único do modelo, K, não contribui para um erro menor em classificação. Eles concluem que o kNN alcança melhor desempenho em tarefas de classificação quando há poucas classes, e com $3 \leq K \leq 14$. Além disso, os autores encontram um desempenho melhor utilizando distância Minikowski, com a distância Manhattan em um segundo lugar não muito distante. BATISTA e SILVA (2009), por sua vez, fazem experimentos com 31 *datasets*, dos quais 28 são da UCI, e

chegam a algumas conclusões relevantes. Primeiramente, não houve diferença significativa entre as funções de distância testadas – Euclidiana, Manhattan e VDM. Além disso, os autores apontam um aumento de desempenho conforme K aumenta, até um máximo no intervalo $5 \leq K \leq 11$. Finalmente, os autores também indicam que o uso de peso como inverso da distância desempenha significativamente melhor que as outras funções de peso testadas.

Portanto, nos experimentos modificados, não foi realizada otimização de parâmetros para kNN. Em vez disso, foram usados os seguintes parâmetros fixos: $K = 7$, função de peso inverso à distância, e distância Manhattan. A junção de K relativamente alto e penalização de distância com a função de peso atuam como fatores de regularização para o modelo, evitando o *overfitting*.

SVM SYARIF *et al.* (2016) estudam o impacto de otimização de parâmetros para SVM de diversos *kernels*, comparando parâmetros padrão com otimização via Grid Search e Genetic Algorithm. Os autores concluem que o desempenho de SVM é significativamente melhor com otimização de parâmetros, ainda que alguns fatores a dificultem – como o intervalo irrestrito para os parâmetros C e *gamma*. ELGELDAWI *et al.* (2021), similarmente, estudam o impacto de otimização de parâmetros de classificadores para *sentiment analysis* entre diversos algoritmos, sendo SVM um deles. Inicialmente, os autores experimentam com parâmetros padrão – e SVM já alcança o melhor desempenho dos modelos avaliados. Contudo, dentre os métodos de busca de parâmetros testados, tanto Grid Search quanto Random Search encontram parâmetros de SVM que obtêm um nível ainda mais alto de acurácia ($> 95\%$).

Como há melhora significativa no desempenho de SVM via *parameter tuning*, as seguintes otimizações foram escolhidas: o parâmetro C, que atua como fator de regularização do modelo, é selecionado a partir da distribuição Log-Uniforme em $[10^{-2}, 10^{1}]$; após alguns testes de refinamento, os intervalos escolhidos para a distribuição Log-Uniforme do parâmetro *gamma*, para *kernel* RBF, foram $[10^{-9}, 10^{3}]$ e, para o *kernel* polinomial, $[10^{-14}, 10^{-3}]$; finalmente, o *kernel* polinomial também possui parâmetro *degree*, buscado uniformemente em $\{2, 3, 4\}$.

Há um terceiro kernel de SVM usado neste trabalho, intitulado "pré-computado". ZHANG *et al.* (2017), em experimentos similares de otimização de parâmetros, indicam que não há necessidade de realizar *parameter tuning* para esse *kernel*, e enfatizam o tempo economizado por isso. De fato, em testes preliminares no desenvolvimento deste trabalho, a otimização de parâmetros no *kernel* pré-computado – variando C em distribuição Log-Uniforme em $[10^{-2}, 10^{1}]$ – demorou na escala de horas, em contraste com poucos mi-

nutos de outros modelos. Assim, manteve-se este *kernel* sem otimização de parâmetros.

GBDT Foram encontrados poucos trabalhos na literatura lidando com diferenças de desempenho de GBDT com otimização de parâmetros. HANCOCK e KHOSHGOFTAAR (2021) estudam, especificamente, o impacto da otimização de parâmetros no desempenho de classificadores para grandes conjuntos de dados desequilibrados. Eles encontram que, sim, há uma melhora significativa no desempenho ao otimizar implementações de GBDT testadas sob essas condições.

A partir dos indícios de melhora de desempenho devido à otimização de parâmetros, os seguintes cinco foram escolhidos para serem otimizados: *learning_rate*, com uma distribuição Uniforme em $[0.01, 1.01]$,[3] que atua como fator de regularização; *n_estimators*, com valor em $\{50, 100, 150, 200, 250\}$; *subsample*, com distribuição Uniforme em $[0.2, 0.99]$ (valores < 1 resultam em Gradient Boosting estocástico); *max_depth*, com valor em $\{3, 6, 12, 24, 48\}$; e *max_features*, com valor em $\{N, \sqrt{N}\}$, em que N é a quantidade de *features*, ou dimensões, dos dados.

Naive Bayes ELGELDAWI *et al.* (2021), como mencionado anteriormente, realizam experimentos para comparar o desempenho de algoritmos com e sem otimização de parâmetros; Naive Bayes é um dos classificadores testados. Seu desempenho com parâmetros padrão é o pior dentre os algoritmos avaliados, situando-se por volta de 70% de acurácia. Mesmo após a otimização de parâmetros, o desempenho não fica tão melhor: NB alcança acurácias de até 78%. Contudo, há melhora – e o trabalho lida com *sentiment analysis* de texto em árabe, um nicho que não necessariamente reflete a qualidade do algoritmo em todas as tarefas.

Uma dificuldade é determinar o intervalo de valores a buscar para a otimização. A implementação do algoritmo utiliza essencialmente um único parâmetro, *var_smoothing*. Nos experimentos de ELGELDAWI *et al.* (2021), os melhores valores encontrados para ele recaem nas ordens de grandeza de $\{10^{-4}, 10^{-5}, 10^{-7}\}$. Na implementação da biblioteca usada, o valor padrão para o parâmetro é de 10^{-9}. Assim, adicionando uma margem de segurança, nos experimentos deste trabalho utilizou-se uma distribuição Log-Uniforme entre $[10^{-9}, 10^{0}]$ para o parâmetro *var_smoothing*.

[3] O valor de "1.01" decorreu de um erro de implementação. A distribuição Uniforme usada, da biblioteca SciPy, é 'stats.uniform(a, b)'. Ela não retorna uma distribuição Uniforme em $[a, b]$, como era esperado, mas sim $[a, a + b]$. Uma desatenção causou o uso de 'stats.uniform$(0.01, 1)$', na expectativa de uma distribuição Uniforme em $[0.01, 1]$, mas ao invés disso representando distribuição Uniforme em $[0.01, 1.01]$.

4.1.2 Métricas de desempenho

Como descrito na seção 3.6.1, as métricas de MSE e LogLoss tiveram relações essencialmente lineares, nos experimentos iniciais, quando comparadas com acurácia. Assim, no intuito de substituí-las por métricas distintas e individualmente interessantes, elas foram removidas.

Em uma tentativa de introduzir uma nova métrica de desempenho mais significativa e robusta que as comumente utilizadas, CARUANA e NICULESCU-MIZIL (2004) realizam experimentos com, dentre outras, a inédita "SAR" (*Squared error, Accuracy and ROC area*), que combina acurácia, AUC e RMSE (*Root Mean Squared Error*). Contudo, após a realização de experimentos, os autores concluem que RMSE, por si só, já é uma métrica muito bem equilibrada, a ponto de tornar a introdução de SAR desnecessária. Portanto, a nova métrica de desempenho escolhida para os experimentos modificados foi RMSE.

Outro problema encontrado nos experimentos iniciais foi a falta de imparcialidade do tempo de treino. Alguns algoritmos, como kNN, não possuem uma etapa de "treino", por dizer; assim, houve uma disparidade enorme entre os classificadores nessa métrica. Dessa forma, com o intuito de tornar as avaliações mais justas, o tempo de teste também passou a ser registrado, além da soma dos dois, intitulada "tempo total". A Tabela 4.1 lista as métricas de desempenho finais, alteradas para a experimentação modificada.

Tabela 4.1: Métricas de desempenho escolhidas para experimentos modificados

#	**Método**	**Implementação**	**Referências**
1	Acurácia / Overall Success Rate	Via scikit-learn: 'sklearn.metrics.accuracy_score'	KING *et al.* (1995); FERRI *et al.* (2009); LABATUT e CHERIFI (2012)
2	F-score / F-measure	Via scikit-learn: 'sklearn.metrics.f1_score'	LABATUT e CHERIFI (2012)
3	Root Mean Squared Error (RMSE)	Via scikit-learn (1.4+): 'sklearn.metrics. root_mean_squared_error'	CARUANA e NICULESCU-MIZIL (2004)
4	Area Under ROC Curve (AUC)	Via scikit-learn: 'sklearn.metrics.roc_auc_score'	FERRI *et al.* (2009)
5	Tempo de treino	Via Python 3.7+: 'time.monotonic_ns'	KING *et al.* (1995)
6	Tempo de teste	Via Python 3.7+: 'time.monotonic_ns'	KING *et al.* (1995)
7	Tempo total	Soma de tempo de treino e teste.	—

4.2 Experimentos

Para cada parâmetro do gerador de dados, criou-se um intervalo de valores interessantes a serem testados. Então, cinco pontos igualmente espaçados nesses intervalos foram escolhidos,[4] e foram realizados dez testes em cada um dos pontos, para cada configuração de parâmetros adicionais testados (Tabela 4.3).

Tabela 4.3: Variação de parâmetros realizada nos experimentos revisados

Parâmetro	**Intervalo**	**Parâmetros adicionais testados**
α	5 pontos em $[0.0001, 1]$: $\{0.0001, 0.250075, 0.50005, 0.750025, 1\}$	unbal = $\{0, 0.2, 0.5, 0.7\}$ noise = $\{0, 5, 20, 50\}$
unbal	5 pontos em $[0.1, 0.7]$: $\{0.1, 0.25, 0.4, 0.55, 0.7\}$	noise = $\{0, 5, 20, 50\}$ alpha = $\{0.0001, 0.001, 0.01, 0.1, 1\}$
noise	5 pontos em $[5, 50]$: $\{5, 16.25, 27.5, 38.75, 50\}$	unbal = $\{0, 0.1, 0.4, 0.7\}$ alpha = $\{0.0001, 0.001, 0.01, 0.1, 1\}$

Similarmente ao procedimento para os primeiros experimentos, o desempenho de cada teste foi registrado em um banco de dados SQLite. Nos experimentos modificados, também foi realizado *parameter tuning* para alguns classificadores, como descrito anteriormente; os valores ótimos para parâmetros encontrados pela otimização também foram registrados. Ao final dos experimentos, o banco de dados continha 30,800 linhas no total.

4.3 Resultados

Os resultados dos experimentos modificados estão descritos nas seções seguintes, separados por áreas de interesse. Uma análise geral dos resultados do trabalho está presente no capítulo 6.

4.3.1 Tempo

Após as alterações nos experimentos, passou-se a registrar três medidas de tempo: tempo de treino (Figura 4.1), de teste (Figura 4.2), e total (Figura 4.3). Enfatiza-se que a escala dos eixos verticais são logarítmicas.

Pode-se comparar a Figura 4.1, de tempo de treino, com a Figura 3.12, de medida equivalente nos primeiros experimentos. A ordem dos modelos em comum (i.e. sem *parameter tuning*) manteve-se relativamente inalterada. Na ocasião, separou-se os modelos em três grupos: lentos (RF e GBDT), meio-termo (SVMs) e rápidos (kNN

[4] Via 'numpy.linspace'.

e NB), com a exceção do SVM de *kernel* pré-computado, que abrangia múltiplas categorias devido à sua alta variância. Aqui, de forma similar, pode-se quebrar os resultados em quatro categorias, em ordem decrescente de tempo: muito lento (GBDT PT, isto é, com *parameter tuning*), lentos (SVMs RBF PT e poli PT, RF, NB PT, GBDT), médios (SVMs RBF e poli) e rápidos (kNN e NB). Novamente, a alta variância do SVM de *kernel* pré-computado dificulta sua alocação a uma única categoria, mas ele se posicionaria entre muito lento e lento.

Contudo, o gráfico também apresenta os modelos com otimização de parâmetros que, como esperado, possuem tempos de treino maiores que suas versões *out-of-the-box*. O pior caso dessa situação surge com o modelo GBDT com otimização, que alcança tempos quatro ou cinco ordens de grandeza mais lentos que os melhores modelos. Em seguida, estão os SVMs otimizados; e, finalmente, o NB otimizado, que possui um tempo comparável a RF.

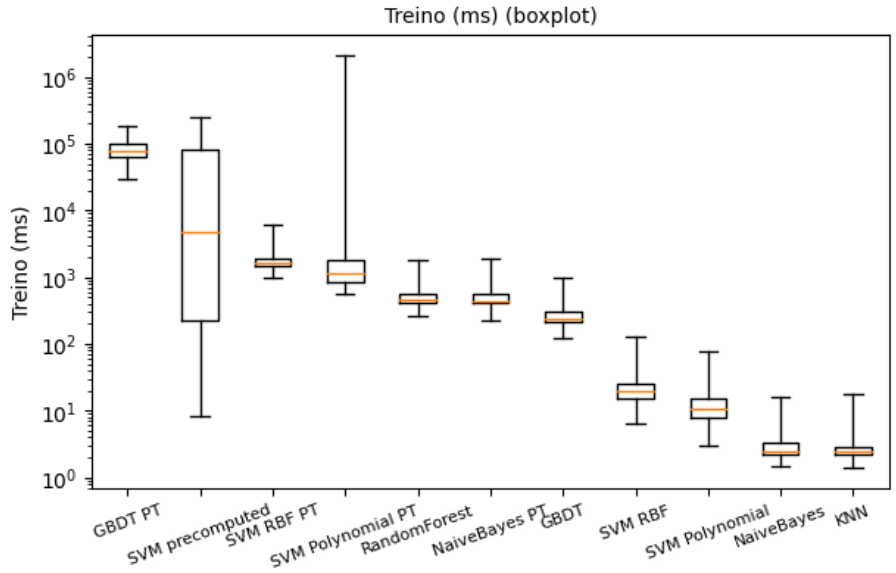


Figura 4.1: Gráfico *boxplot* de classificadores medidos por tempo de treino, em ordem decrescente de tempo. A escala do eixo vertical é logarítmica.

Os tempos de teste dos experimentos revisados (Figura 4.2) foram, em geral, mais curtos do que os tempos de treino, mas eles evidenciam a diferença de comportamento dos diversos modelos.

Por exemplo, kNN, um dos modelos mais velozes em tempo de treino, posicionou-se como o quarto pior em tempo de teste. A diferença, aqui, não é tão proeminente – todos possuem tempos razoavelmente similares – mas, ainda assim, o kNN perde sua posição de liderança.

A lentidão relativa do kNN é esperada: durante a etapa de treino, ele somente guarda os pontos; quando precisa classificar, aí sim o algoritmo realiza maior esforço computacional, avaliando a proximidade do ponto a ser classificado com os pontos

armazenados durante o treino. Esse é um processo lento que, dependendo da implementação e da quantidade de pontos de treino, pode acabar com a vantagem de um curto tempo de treino.

Por sua vez, NB mantém-se como o mais rápido, tanto *out-of-the-box* quanto com otimização de parâmetros. Na realidade, todos os modelos com parâmetros otimizados posicionaram-se próximos às suas versões originais, com a exceção de GBDT, cuja versão otimizada teve variância significativamente maior.

E, no outro lado do espectro, RF aparece como o modelo mais lento nos testes, a pelo menos uma ordem de grandeza de distância de NB. Assim, pode-se separar os classificadores, novamente, em categorias: na mais lenta, somente RF; na de desempenho médio, SVMs, GBDT e kNN; e na mais rápida, NB.

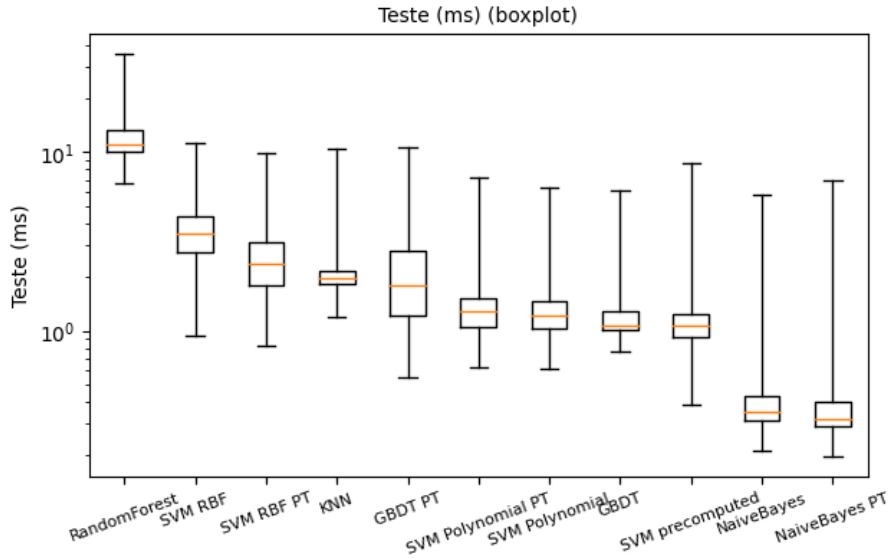


Figura 4.2: Gráfico *boxplot* de classificadores medidos por tempo de teste, em ordem decrescente de tempo. A escala do eixo vertical é logarítmica.

Finalmente, há a métrica de tempo total (Figura 4.3), o tempo de treino somado ao de teste. Devido às diferenças de escala – o tempo de treino frequentemente atingiu tempos acima de 1 segundo (1,000 ms), enquanto o tempo de teste não chega aos 100 ms – os resultados são muito similares aos de tempo de treino.

Entretanto, notavelmente, o posicionamento do kNN muda. Se, antes, ele, ainda que por uma pequena margem, possuía mediana menor que a do NB, após a soma dos tempos de teste suas posições trocam, com NB com parâmetros *out-of-the-box* sendo o modelo mais rápido para ambas as tarefas. Em contraste, o classificador GBDT com otimização de parâmetros posiciona-se como o mais lento por uma boa margem, a quatro ou cinco ordens de grandeza de distância de NB.

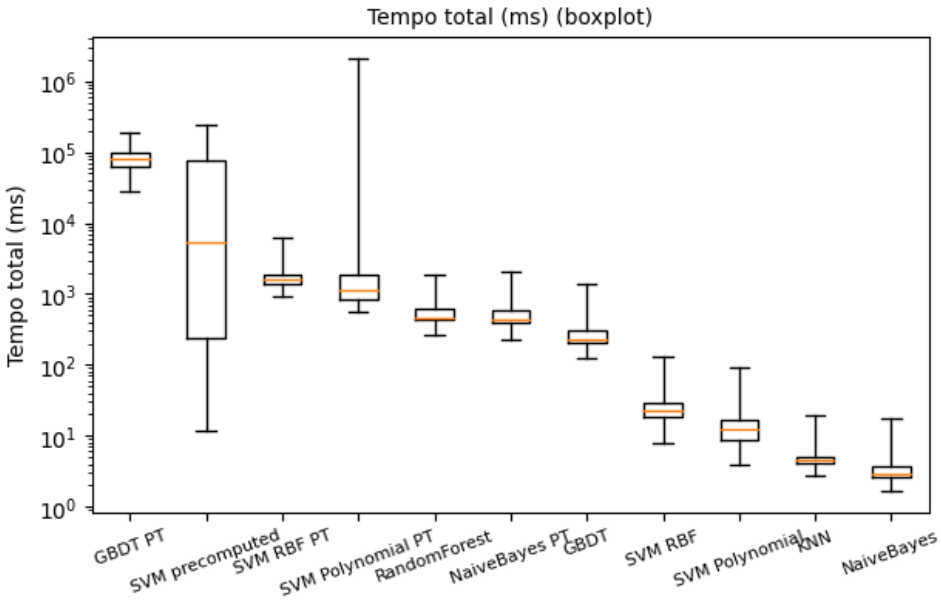


Figura 4.3: Gráfico *boxplot* de classificadores medidos por tempo total (treino + teste), em ordem decrescente de tempo. A escala do eixo vertical é logarítmica.

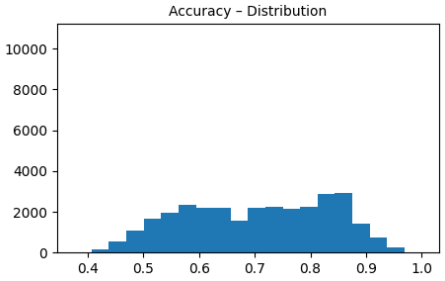
4.3.2 Correlação tempo vs. desempenho

Uma nova possibilidade interessante é comparar o tempo gasto por algoritmo e seu desempenho. Isto é, um algoritmo com mais tempo de treino e teste obtém resultados melhores? Nos testes iniciais, somente o tempo de treino foi medido, o que dificulta essa avaliação. Contudo, com a medição de tempo de teste, é possível realizar uma análise mais aprofundada. A seguir, verifica-se a correlação entre as diversas métricas de desempenho e o tempo total de execução – isto é, tempo de treino somado ao tempo de teste – de cada modelo.

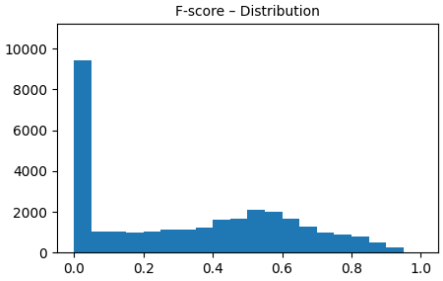
O teste de correlação de Pearson é comumente utilizado para detectar relações lineares em dados. No entanto, ele pressupõe uma distribuição normal (ou quase normal) dos dados, e pode ser sensível a distribuições não-normais – vide KOWALSKI (1972). Como não há garantia de normalidade na distribuição dos valores encontrados – e, experimentalmente, eles se comportam de forma não-normal, vide Figura 4.4 – as correlações a seguir foram calculadas utilizando o teste de correlação de Kendall.[5]

O teste de Kendall é um teste não-paramétrico – ou seja, não presume uma distribuição específica nas variáveis que examina. Além disso, o teste aponta relações não somente lineares, mas monotônicas entre duas variáveis. Assim, é uma escolha ideal para a verificação de correlação desejada.

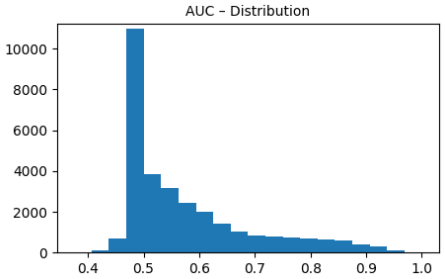
[5]Via 'scipy.stats.kendalltau'.



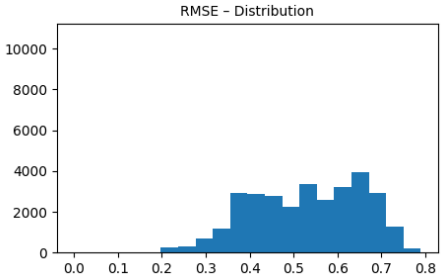
(a) Distribuição de acurácia encontrada experimentalmente.



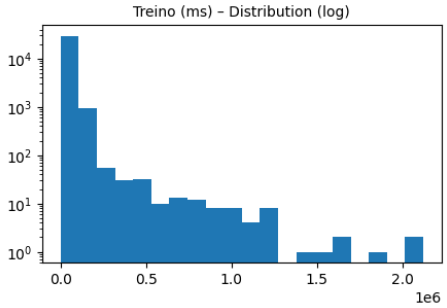
(b) Distribuição de F-score encontrada experimentalmente.



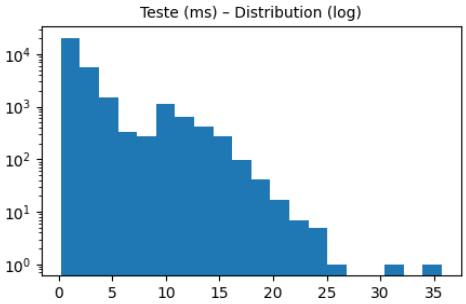
(c) Distribuição de AUC encontrada experimentalmente.



(d) Distribuição de RMSE encontrada experimentalmente.



(e) Distribuição de tempo de treino encontrada experimentalmente. O eixo vertical é logarítmico.



(f) Distribuição de tempo de teste encontrada experimentalmente. O eixo vertical é logarítmico.

Figura 4.4: Distribuição de valores encontrados experimentalmente entre as métricas de desempenho utilizadas.

As tabelas a seguir (Tabela 4.5, Tabela 4.7) apresentam os resultados encontrados para os testes de correlação entre as diferentes métricas de desempenho e tempo total de execução. Estão enfatizadas as três maiores correlações em cada coluna (em valores absolutos), além de ocorrências de p-value > 0.05 – isto é, casos em que não encontra-se significância estatística, com 95% de confiança, da ocorrência de correlação entre os parâmetros. Em termos práticos, quanto maior o *p-value*, maior a chance de se obter os mesmos resultados por acaso, em distribuições sem correlações intrínsecas.

Do total de 44 correlações calculadas, oito (18.2%) estão fora do nível de confiança de 95%. O restante das métricas de desempenho por modelo provavelmente

possui, sim, alguma correlação com o tempo total de execução. Contudo, grande parte das correlações são fracas – das 36 que possuem confiança acima de 95%, 18 (50%) possuem correlações abaixo de 10% em valor absoluto.

Tabela 4.5: Correlação entre acurácia, F-score e tempo total de execução

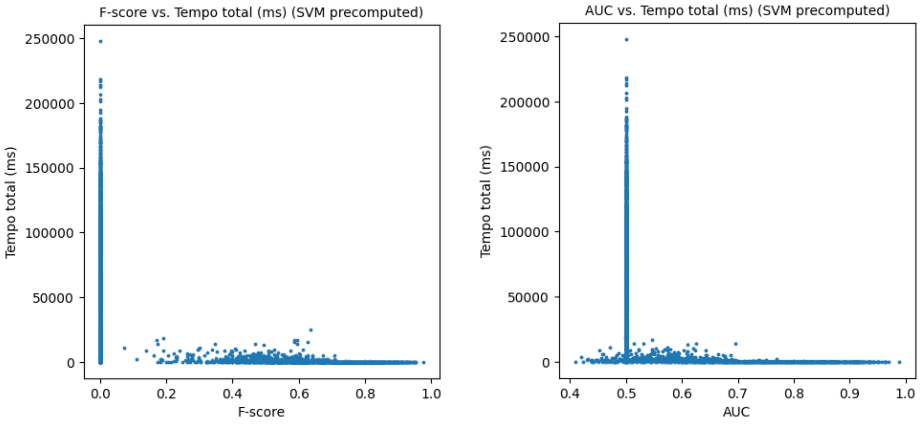
Modelo	**Acurácia vs. tempo**		**F-score vs. tempo**	
	Kendall	***p-value***	**Kendall**	***p-value***
kNN	+0.027	**0.031**	−0.027	0.035
RF	−0.162	**0.000**	*−0.020*	*0.119*
NB	*+0.019*	*0.145*	−0.083	0.000
NB PT	*+0.000*	*0.994*	+0.029	0.029
GBDT	−0.026	0.038	+0.031	0.014
GBDT PT	−0.154	0.000	**−0.340**	**0.000**
SVM pré-comp.	−0.056	0.000	**−0.624**	**0.000**
SVM RBF	−0.166	0.000	**−0.276**	**0.000**
SVM RBF PT	**−0.208**	**0.000**	+0.042	0.002
SVM pol.	**−0.400**	**0.000**	*−0.003*	*0.847*
SVM pol. PT	**−0.294**	**0.000**	−0.027	0.035

Tabela 4.7: Correlação entre AUC, RMSE e tempo total de execução

Modelo	**AUC vs. tempo**		**RMSE vs. tempo**	
	Kendall	***p-value***	**Kendall**	***p-value***
kNN	*+0.003*	*0.811*	−0.028	0.031
RF	−0.079	0.000	+0.162	0.000
NB	−0.052	0.000	*−0.019*	*0.145*
NB PT	+0.042	0.001	*−0.000*	*0.994*
GBDT	+0.031	0.014	+0.026	0.038
GBDT PT	**−0.324**	**0.000**	+0.154	0.000
SVM pré-comp.	**−0.587**	**0.000**	+0.056	0.000
SVM RBF	**−0.282**	**0.000**	+0.166	0.000
SVM RBF PT	*−0.005*	*0.683*	**+0.208**	**0.000**
SVM pol.	−0.051	0.000	**+0.400**	**0.000**
SVM pol. PT	+0.089	0.000	**+0.294**	**0.000**

Além disso, alguns outros fatores contribuem para as correlações mais altas. Por exemplo, nas duas maiores (de −0.624 e −0.587, em F-score e AUC vs. tempo, ambas para o SVM com *kernel* pré-computado), ao *plotar* os valores (Figura 4.5), encontra-se que a correlação é fruto de anomalias. Em ambos os casos, o SVM pré-computado teve desempenho muito fraco – com muitos F-scores iguais a 0 e AUCs iguais a 0.5. Assim, calcula-se uma correlação negativa. O motivo por trás dessas anomalias é discutido na seção 6.1.

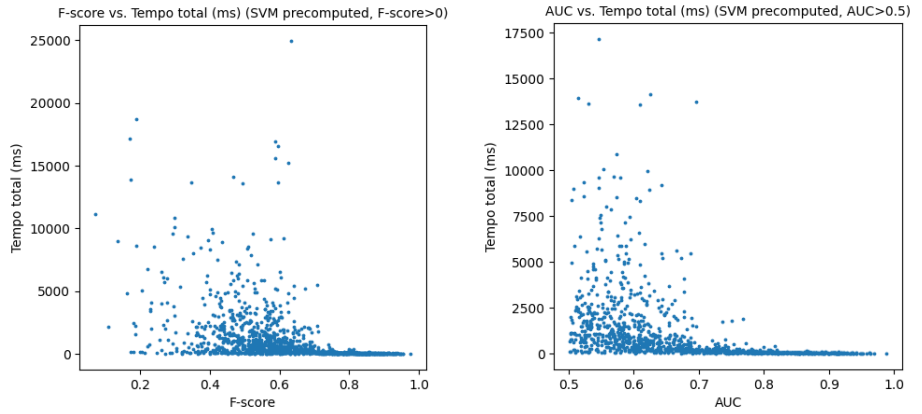
Se os resultados forem restritos a estritamente F-score > 0 e AUC > 0.5, respectivamente, como demonstrado na Figura 4.6, os valores calculados para a correlação



(a) Distribuição de acurácia encontrada experimentalmente.

(b) Distribuição de F-score encontrada experimentalmente.

Figura 4.5: Gráfico das duas maiores correlações, calculadas a -0.624 e -0.587, para F-score e AUC vs. tempo total, ambas no modelo de SVM com kernel pré-computado.



(a) Distribuição de F-score encontrada experimentalmente.

(b) Distribuição de F-score encontrada experimentalmente.

Figura 4.6: Gráfico das duas maiores correlações, F-score e AUC vs. tempo total, ambas no modelo de SVM com *kernel* pré-computado, com modificações: F-score está restrito a valores não-nulos, e AUC está restrito a valores acima de 0.5.

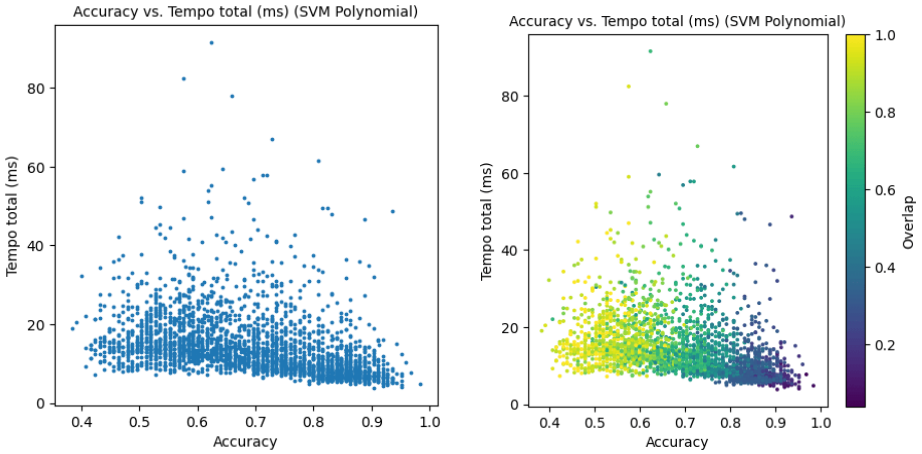
se tornam -0.48 e -0.605. Isto é, a magnitude da correlação diminui para o F-score vs. tempo, mas aumenta para AUC vs. tempo.

Uma situação menos inusitada se encontra no terceiro lugar por magnitude de correlação: acurácia vs. tempo para o SVM com *kernel* polinomial, com correlação de aproximadamente -0.4. A Figura 4.7(a) mostra o *scatter plot* dos valores encontrados. Percebe-se que, de fato, há uma tendência: quanto menor o tempo de execução, melhor a acurácia alcançada.

Isso, claro, não significa que há, necessariamente, uma relação de causalidade entre as duas métricas. Uma explicação mais simples e razoável é a existência de

um terceiro fator causador de ambos os efeitos. Por exemplo, se problemas mais "fáceis" gastam menos tempo para serem solucionados, eles podem simultaneamente alcançar acurácias mais altas.

E, de fato, isso pode ser verificado na Figura 4.7(b), na qual os pontos são coloridos em função da sobreposição calculada dos dados. Há uma clara relação: quanto mais sobrepostos estão os dados (cor mais clara), menor a acurácia, e também maior o tempo de execução; inversamente, quanto menor a sobreposição, mais "fácil" é o problema, maior a acurácia e menor o tempo de execução. Em seu trabalho, OH (2011) corrobora a relação de sobreposição vs. desempenho, apesar de não medir tempo de execução.



(a) Acurácia vs. tempo total.

(b) Acurácia vs. tempo total vs. sobreposição, com este último indicado por um espectro de cores.

Figura 4.7: Relação entre acurácia e tempo total de execução para o modelo SVM com *kernel* polinomial. O teste de Kendall indica uma correlação de aproximadamente -0.4. Isto é, quanto maior a acurácia, menor o tempo de execução.

4.3.3 Correlação entre métricas

Nos experimentos iniciais, foram percebidas similaridades entre algumas métricas de desempenho usadas. LogLoss e MSE tinham relações essencialmente lineares com acurácia. Para evitar a redundância nas métricas, foram feitas modificações em quais seriam utilizadas, removendo LogLoss e MSE, e adicionando RMSE, além de tempos de teste e total.

Após os testes adicionais, é interessante conferir se as métricas ainda possuem alguma correlação. Pela Figura 4.8, percebe-se que RMSE possui uma relação monotônica com acurácia, mas não linear; então, a troca foi bem sucedida.

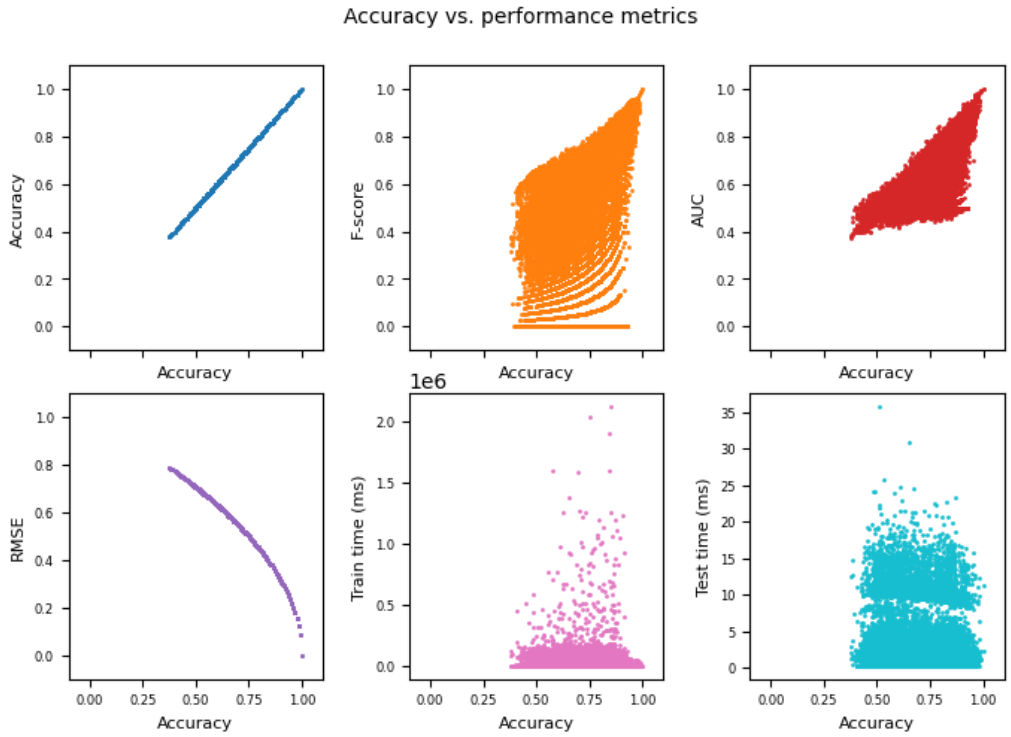


Figura 4.8: Medições de cada métrica de desempenho utilizada vs. acurácia.

4.3.4 Desempenho entre modelos

O objetivo inicial deste trabalho era buscar por diferenças significativas no desempenho de modelos de classificação, com o intuito de descobrir vantagens ou desvantagens de algoritmos individuais. Para tal, é necessário comparar o desempenho de cada um diretamente. As figuras a seguir apresentam os resultados medidos para cada uma das métricas de desempenho selecionadas: acurácia, F-score, AUC e RMSE.

Para acurácia (Figura 4.9), há tão pouca diferença entre os classificadores que não é possível elencar um como sendo melhor ou pior que os outros. Em situações nas quais não foi aplicado ruído e desequilíbrio nos dados (Figura 4.10), similarmente, todos os classificadores se comportam de forma muito parecida, com a notável exceção do SVM com *kernel* polinomial e otimização de parâmetros: sua mediana recai bem abaixo das dos outros modelos.

Verificando o desempenho medido por F-score (Figura 4.11), há novamente sobreposição de todos os *boxplots*, indicando que não há diferença significativa entre o desempenho de cada classificador.

Contudo, conforme o gráfico de violino permite notar, a distribuição dos resultados não é igual entre classificadores. Em muitos dos classificadores, há uma grande aglutinação próxima de F-score = 0. Sem matrizes de confusão ou outras métricas, não é possível determinar com absoluta certeza a causa desse agrupamento; vide

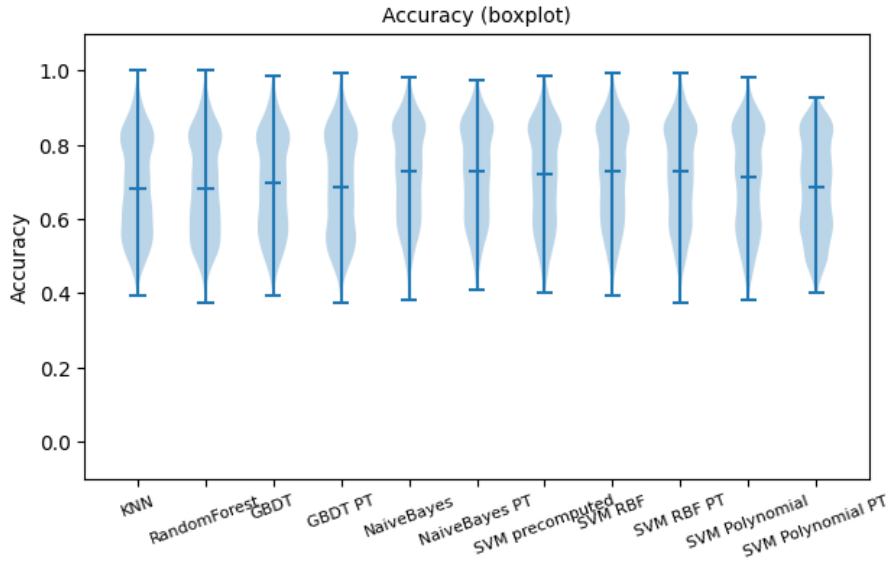


Figura 4.9: Desempenho de cada modelo medido, por acurácia (maior é melhor).

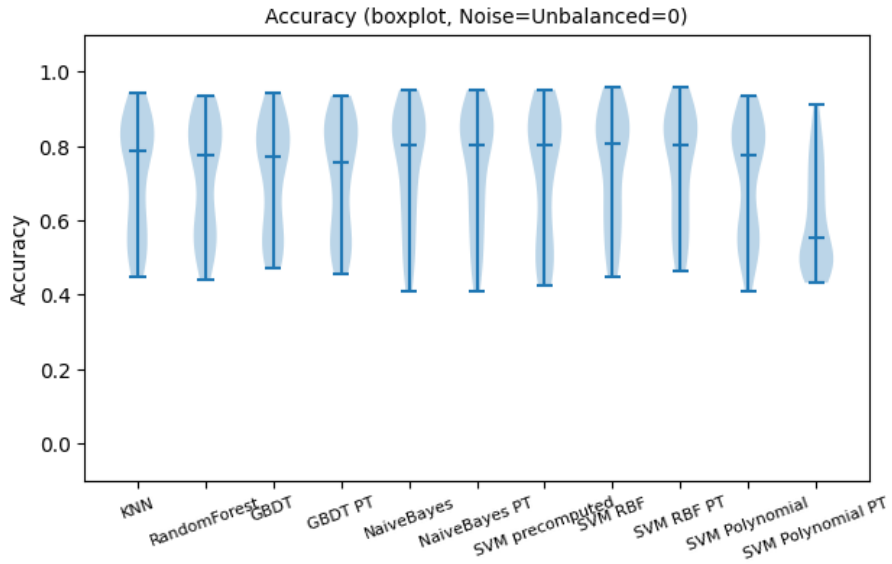


Figura 4.10: Desempenho de cada modelo medido, por acurácia (maior é melhor), sem ruído ou classes desbalanceadas.

seção 6.1. Contudo, ele provavelmente se dá devido a classificadores que falharam em aprender os dados de treino, e retornaram exclusivamente classes negativas para todos os pontos. Quando medido por F-score, isso é um desempenho de 0.

Para expandir nessa hipótese, pode-se contar ocorrências de F-score = 0, como na Figura 4.12. Percebe-se que alguns classificadores são mais propensos a esse tipo de erro (SVMs e NB) independentemente de otimização de parâmetros. Os outros, kNN, RF e GBDT, mostram-se mais capazes de classificar de forma mais equilibrada. Há mais de dez vezes mais ocorrências desse problema quando comparando o pior desempenho, pelo SVM de *kernel* polinomial com parâmetros otimizados, com os

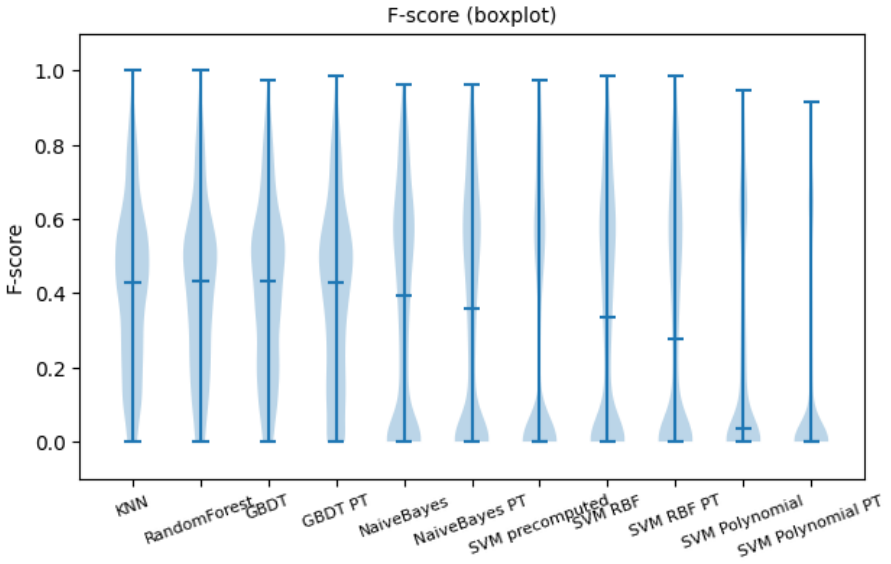


Figura 4.11: Desempenho de cada modelo, medido por F-score (maior é melhor).

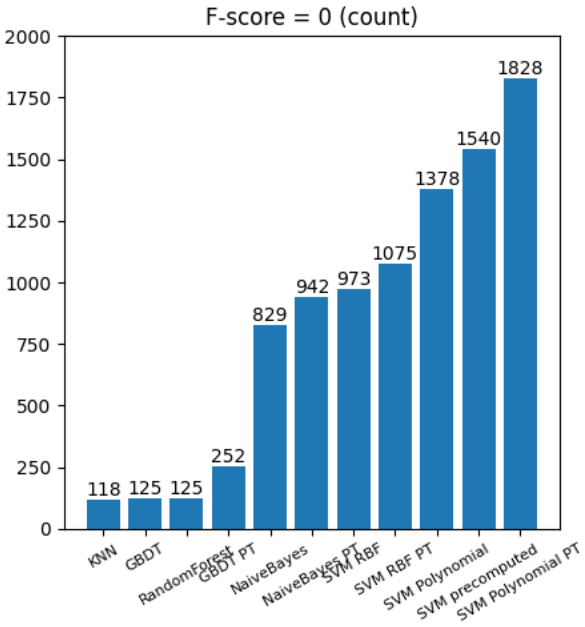


Figura 4.12: Ocorrências de F-score = 0 agrupadas por modelo.

três melhores, por kNN, RF e GBDT.

No interesse da completude, os desempenhos medidos por AUC e RMSE estão expostos na Figura 4.13 e na Figura 4.14, respectivamente. A mesma situação encontrada anteriormente ocorre aqui: os classificadores se comportam muito similarmente, ainda que com pequenas diferenças em medianas e variâncias.

Pode-se repetir a análise feita para F-score, mas contando ocasiões em que classificadores obtiveram AUC = 0.5 (Figura 4.15(a)). Um desempenho de 0.5 em AUC indica nenhum aprendizado, equivalente a usar cara e coroa de uma moeda para classificar. Além disso, é possível contar ocorrências de AUC $<$ 0.5 (Figura 4.15(b)),

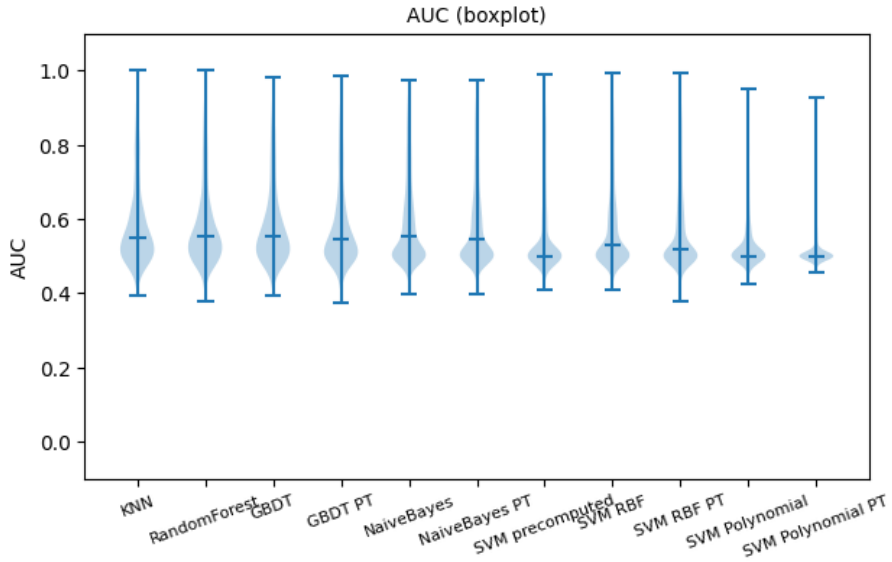


Figura 4.13: Desempenho de cada modelo, medido por AUC (maior é melhor).

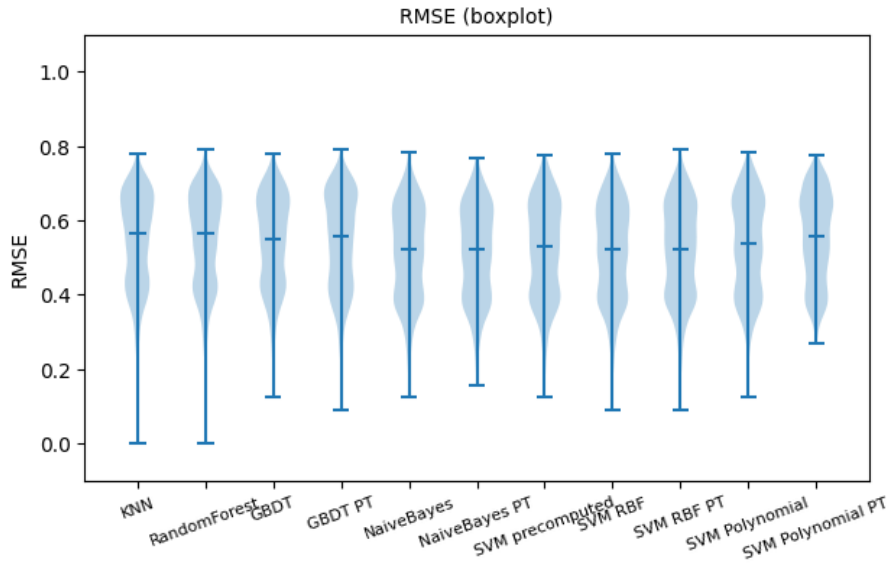
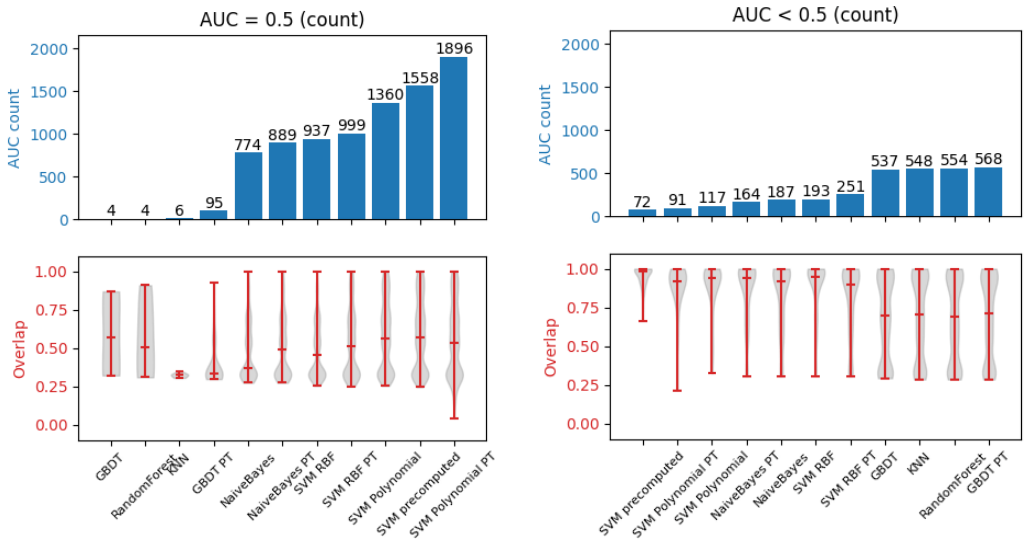


Figura 4.14: Desempenho de cada modelo medido por RMSE (menor é melhor).

indicando um desempenho pior que aleatório.

Na primeira situação, kNN, GBDT e RF mantêm-se com menos casos, enquanto SVM com *kernels* polinomial (com e sem otimização de parâmetros) e pré-computado mantém-se com mais casos – com uma diferença de quase 500 vezes entre o menor e o maior.

Para AUC < 0.5, a situação se inverte: kNN, GBDT (com e sem otimização) e RF aparecem muito próximos, com maior quantidade de ocorrências. Em contraste, as menores quantidades são encontradas em SVM com *kernel* pré-computado e polinomial, com e sem otimização. No entanto, é importante notar que as maiores quantidades de instâncias de AUC < 0.5 ainda são aproximadamente três vezes



(a) Ocorrências de AUC = 0.5 agrupadas por modelo, e sobreposição de classes calculada para os dados dessas ocorrências.

(b) Ocorrências de AUC < 0.5 agrupadas por modelo, e sobreposição de classes calculada para os dados dessas ocorrências.

Figura 4.15: Contagem de ocorrências de valores de AUC.

menores que as maiores quantidades de AUC = 0.5.

A sobreposição das classes, calculada para cada instância contada pelos gráficos, também está presente em ambas as figuras. Ela aponta uma relação interessante: erros não estão restritos a sobreposições altas, como poderia ser esperado, ainda que estas também existam para casos de AUC < 0.5; ao invés disso, percebe-se um aumento de ocorrências de sobreposição em torno de 0.3 nos maiores casos de AUC baixo. Isso pode se dar por alguma configuração particular, de difícil classificação, com esse *overlap*; ou, mais simplesmente, por uma maior frequência desse valor de sobreposição – vide seção 4.3.7.

4.3.5 Parâmetros otimizados

Outra informação relevante que pode ser extraída dos experimentos são os valores otimizados para os parâmetros dos classificadores que passaram por *parameter tuning*. É claro que estes se aplicam somente às distribuições de dados em que foram testados, e não a qualquer *dataset*. Assim, não pode-se tirar conclusões de aptidão dos valores escolhidos como padrão para a implementação, que devem focar em um bom desempenho para a maior gama de dados possível.

A Figura 4.16 mostra a distribuição de valores otimizados do único parâmetro do modelo NB, *var_smoothing*. Como ele foi escolhido via distribuição Log-Uniforme (entre $[10^{-9}, 10^{0}]$), a escala do eixo horizontal é logarítmica, onde cada valor x representa 10^{x}.

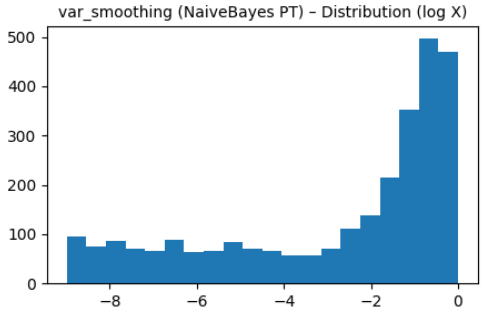
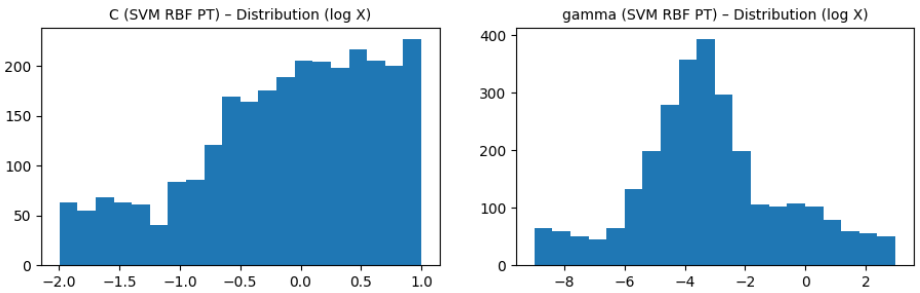


Figura 4.16: Distribuição de valores otimizados para o parâmetro *var_smoothing* do modelo NB. O eixo horizontal está em escala logarítmica; valores indicam 10^x.

Percebe-se que os valores ótimos mais comumente se situam próximos a 10^0, o que é o inverso do esperado: o valor padrão da implementação é 10^{-9}, na ponta oposta da distribuição. Isso, claro, não significa que 10^{-9} é um valor padrão mal escolhido; mas significa que, pelo menos para os dados testados, ele não é o ideal.

Para SVM com *kernel* RBF, são dois parâmetros otimizados: C e *gamma*, escolhidos via Log-Uniforme entre $[10^{-2}, 10^{1}]$ e $[10^{-9}, 10^{3}]$, respectivamente. A Figura 4.17 mostra a distribuição de valores ótimos encontrados pelo Random Search para estes parâmetros. Novamente, ambos os gráficos possuem eixo horizontal em escala logarítmica.



(a) Distribuição de valores otimizados para o parâmetro C.

(b) Distribuição de valores otimizados para o parâmetro *gamma*.

Figura 4.17: Distribuição de valores otimizados para parâmetros de SVM com *kernel* RBF. Os eixos horizontais estão em escala logarítmica; valores indicam 10^x.

Pode-se notar, na Figura 4.17(a), uma tendência a valores maiores e mais próximos de 10^1 para C, que é o valor padrão para para o parâmetro na implementação da biblioteca.

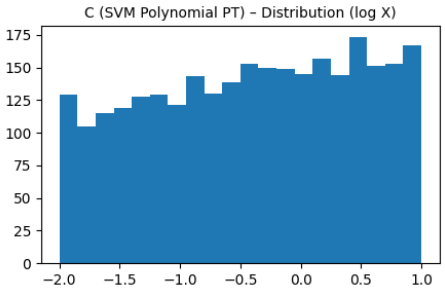
Na Figura 4.17(b), por outro lado, apresenta-se o parâmetro *gamma*, com pico de ocorrência entre 10^{-3} e 10^{-4}. Seu valor padrão na implementação[6] é $\frac{1}{(F \times Var[X])}$ que, no contexto dos dados artificiais gerados nos experimentos, é entre 10^{-3} e 10^{-2};

[6]Disponível em: `https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`. Acesso em 1 nov.2024.

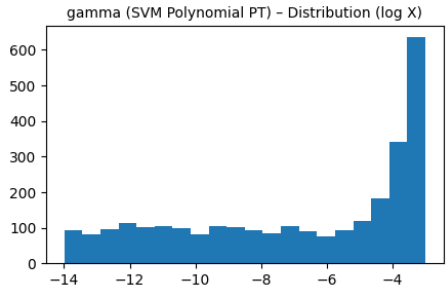
bem próximo dos valores encontrados pela otimização.

No caso do SVM com *kernel* polinomial, novamente há parâmetros C e *gamma*, neste caso escolhidos via Log-Uniforme entre $[10^{-2}, 10^{1}]$ e $[10^{-14}, 10^{-3}]$, respectivamente. Porém, este classificador também possui um parâmetro adicional, *degree*, escolhido uniformemente em $\{2, 3, 4\}$.

Assim como para o *kernel* RBF, o parâmetro *gamma* situa-se majoritariamente nos arredores do valor padrão da implementação, entre 10^{-3} e 10^{-2} (Figura 4.18(b)). Contudo, a distribuição de valores escolhidos para o parâmetro C comporta-se quase de maneira uniforme (Figura 4.18(a)). Isso é um indicativo de que o desempenho do modelo para os dados usados não variou muito independentemente do valor de C – e, portanto, sua escolha não afetou tanto a otimização do modelo.



(a) Distribuição de valores otimizados para o parâmetro C.



(b) Distribuição de valores otimizados para o parâmetro *gamma*.

Figura 4.18: Distribuição de valores otimizados para parâmetros de SVM com *kernel* polinomial. Os eixos horizontais estão em escala logarítmica; valores indicam 10^{x}.

Em termos do parâmetro *degree*, a distribuição também foi bem próxima entre os valores (Tabela 4.9). O valor de 3, que é o valor padrão da implementação, foi aproximadamente 10% mais frequente do que os outros dois valores, o que serve como indicativo – ainda que não muito forte – de que a escolha de valor padrão é adequada.

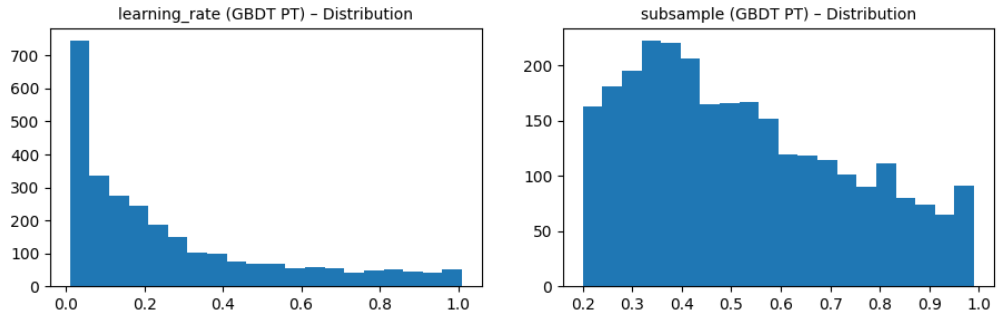
Tabela 4.9: Distribuição de valores otimizados para parâmetro *degree* do SVM Polinomial

Valor	**Ocorrências**	**Percentual**
2	854	30.5%
3	1106	39.5%
4	840	30.0%
Total:	2800	100%

Finalmente, para GBDT, o parâmetro *learning_rate*, escolhido de maneira Uniforme em $[0.01, 1.01]$, teve alta tendência de ser otimizado a valores próximos de

0.01 (Figura 4.19(a)). Isso é um bom sinal, pois seu valor padrão de implementação é 0.1.

Por sua vez, o parâmetro *subsample*, escolhido uniformemente em $[0.2, 0.99]$, teve pico de ocorrências entre 0.3 e 0.4 (Figura 4.19(b)), o que destoa de seu valor padrão de 1. Contudo, é importante ressaltar que somente valores menores que 1 representam *gradient boosting* estocástico, que era o comportamento desejado, então o valor padrão não é tão relevante aqui.



(a) Distribuição de valores otimizados para o parâmetro *learning_rate*.

(b) Distribuição de valores otimizados para o parâmetro *subsample*.

Figura 4.19: Distribuição de valores otimizados para parâmetros de GBDT.

No caso de *n_estimators*, escolhido de maneira uniforme em $\{50, 100, 150, 200, 250\}$, houve maior ocorrência de valores menores (Tabela 4.10); os dois primeiros, 50 e 100, são responsáveis por quase 50% das escolhas. Como referência, este parâmetro possui valor padrão de 100.

O parâmetro *max_depth*, similarmente, teve seu valor padrão de 3 escolhido em 26% das ocasiões (Tabela 4.10), dentre as possibilidades de $\{3, 6, 12, 24, 48\}$. Curiosamente, ignorando 3, há um aumento de frequência gradual conforme o valor aumenta; assim, 48 é o segundo mais ocorrente, escolhido quase 21% do tempo.

Tabela 4.10: Distribuição de valores otimizados para parâmetros *n_estimators*, *max_depth* e *max_features* do GBDT

n_estimators			*max_depth*			*max_features*		
Valor	**Ocorrências**	**Percentual**	**Valor**	**Ocorr.**	**Pct.**	**Valor**	**Ocorr.**	**Pct.**
50	732	26.14%	3	728	26.00%	N	1183	42.25%
100	591	21.11%	6	369	13.18%	$\sqrt{N}$	1617	57.75%
150	540	19.29%	12	552	19.71%	**Total:**	2800	100%
200	509	18.18%	24	571	20.39%			
250	428	15.29%	48	580	20.71%			
Total:	2800	100%	**Total:**	2800	100%			

Por fim, o parâmetro de *max_features*, escolhido uniformemente em $\{N, \sqrt{N}\}$ – onde N representa o número de *features*, ou dimensões, dos dados – teve quase 60% das ocorrências com valor $\sqrt{N}$, em discordância com o valor padrão, N.

4.3.6 Impacto da otimização de parâmetros

Um fator que pode ser testado utilizando os resultados obtidos é a eficácia da otimização de parâmetros. Há valor em fazer *parameter tuning*, mesmo com o tempo adicional que essa etapa incorre no processo? Ou o desempenho dos classificadores não melhora o suficiente para justificar o tempo extra gasto?

Em termos de tempo de execução, a Figura 4.20 compara o desempenho entre pares de modelos, com e sem otimização de parâmetros. Percebe-se, como esperado, que a otimização de parâmetros aumenta significativamente o tempo de execução dos classificadores, devido ao maior tempo de treino. Isso, é claro, é totalmente esperado – mas serve para indicar a magnitude do aumento. Enfatiza-se que o gráfico está em escala logarítmica.

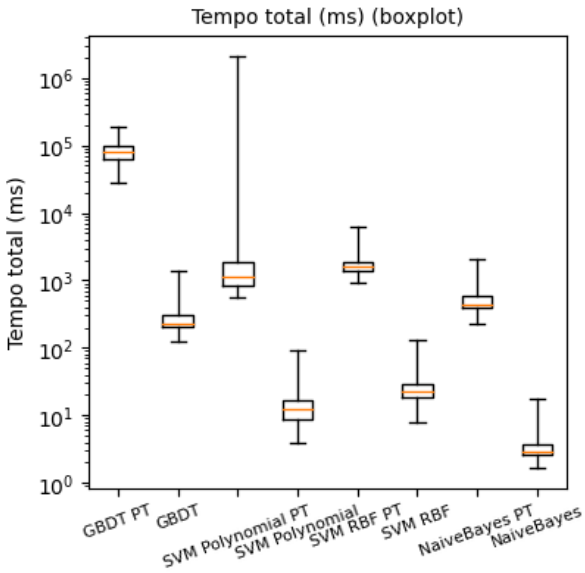


Figura 4.20: *Boxplot* de tempo total de pares de modelos com e sem otimização de parâmetros. O eixo vertical está em escala logarítmica.

Confirmada a diferença significativa de tempo de execução, podemos comparar os algoritmos, então, por seu desempenho medido pelas outras métricas. Os resultados obtidos pelos dois modelos, com e sem otimização, não são independentes – ambos são, em suma, o mesmo algoritmo trabalhando sobre os mesmos dados, somente com alguns parâmetros modificados. Portanto, para medir sua similaridade, faz-se uso de um teste dito "pareado".

A Tabela 4.11 contém os resultados de um teste-t de Student pareado,[7] realizado

[7] Via 'scipy.stats.ttest_rel'.

sobre os desempenhos de cada modelo e de sua versão com parâmetros otimizados, medidos pelas diferentes métricas de desempenho. Em **preto**, estão as instâncias de rejeição da hipótese nula – a hipótese de que não há diferença significativa entre os modelos – com confiança de 95%. Em **cinza**, ocasiões em que não é possível afirmar com tanta certeza que há diferença significativa após a otimização de parâmetros ($p\text{-}value > 0.05$).

Além disso, a tabela também apresenta a diferença da média das métricas de desempenho; i.e., média do modelo com otimização menos média do modelo sem otimização. Para acurácia, F-score e AUC, um valor positivo indica melhora de desempenho com otimização, em média; para RMSE, um valor negativo indica melhora.

Tabela 4.11: Comparação entre modelos antes e depois da otimização de parâmetros

Modelo	**Métrica**	Diferença de média entre modelos Teste-t pareado (p-value)			
		Todos	***noise*, *unbal* = 0**	***noise*, *unbal* > 0**	***overlap* > 0.85**
SVM RBF	Acurácia	−0.00324 0.000	+0.00128 0.545	−0.00301 0.000	−0.00470 0.001
	F-score	−0.01836 0.000	−0.00187 0.572	−0.02150 0.000	−0.04569 0.000
	AUC	−0.00449 0.000	+0.00043 0.844	−0.00505 0.000	−0.00688 0.000
	RMSE	+0.00310 0.000	+0.00006 0.980	+0.00296 0.000	+0.00355 0.001
SVM Pol.	Acurácia	−0.02119 0.000	−0.11568 0.000	−0.01315 0.000	+0.00081 0.601
	F-score	−0.07792 0.000	−0.19312 0.000	−0.06517 0.000	+0.00869 0.186
	AUC	−0.03258 0.000	−0.10944 0.000	−0.02312 0.000	+0.00054 0.667
	RMSE	+0.02145 0.000	+0.10896 0.000	+0.01333 0.000	−0.00058 0.612
NaiveBayes	Acurácia	−0.00019 0.385	−0.00176 0.188	+0.00022 0.373	−0.00074 0.235
	F-score	−0.01679 0.000	−0.00643 0.039	−0.02052 0.000	−0.02393 0.000
	AUC	−0.00395 0.000	−0.00129 0.313	−0.00443 0.000	−0.00197 0.001
	RMSE	+0.00019 0.348	+0.00168 0.119	−0.00022 0.347	+0.00057 0.222
GBDT	Acurácia	−0.00209 0.005	+0.00080 0.893	−0.00131 0.144	−0.00875 0.000
	F-score	−0.01503 0.000	−0.00289 0.752	−0.01830 0.000	−0.00195 0.534
	AUC	−0.00645 0.000	−0.00019 0.975	−0.00669 0.000	−0.00747 0.000
	RMSE	+0.00084 0.195	−0.00179 0.724	+0.00004 0.957	+0.00649 0.000

A Tabela 4.11 separa os resultados em diferentes situações – sem e com ruído e desequilíbrio, nível de sobreposição de dados – com o intuito de facilitar a análise dos modelos. Para todos os testes, na maioria dos casos, há diferença significativa (com pelo menos 95% de confiança) ao introduzir a otimização de parâmetros. Contudo, para todos os casos com diferença significativa, há uma *piora* no desempenho dos algoritmos ao otimizá-los.

Para além disso, em experimentos sem ruído nem desequilíbrio de dados – isto é, dados mais "simples" de classificar –, a significância das diferenças cai e há muito menos instâncias de diferenças com pelo menos 95% de confiança. Isso é esperado, pois, em teoria, se tratam de casos mais "fáceis", em que algoritmos alcançam bons desempenhos mesmo com parâmetros não-refinados.

Finalmente, para dados com sobreposição de classes acima de 85%, a distinção da otimização de parâmetros se mantém, salvo em um caso: o SVM com *kernel* polinomial. Este modelo possui um comportamento curioso, explorado em mais detalhes posteriormente no capítulo 6.

4.3.7 Impacto da sobreposição de classes

Outro fator potencialmente interessante para análise é a sobreposição calculada das classes. É trivial que, quanto menor a sobreposição de classes – e, portanto, quanto mais próximas de linearmente separáveis forem as classes –, melhor o desempenho esperado. A Figura 4.21 apresenta a relação entre sobreposição de classes e desempenho, para todas as métricas escolhidas.

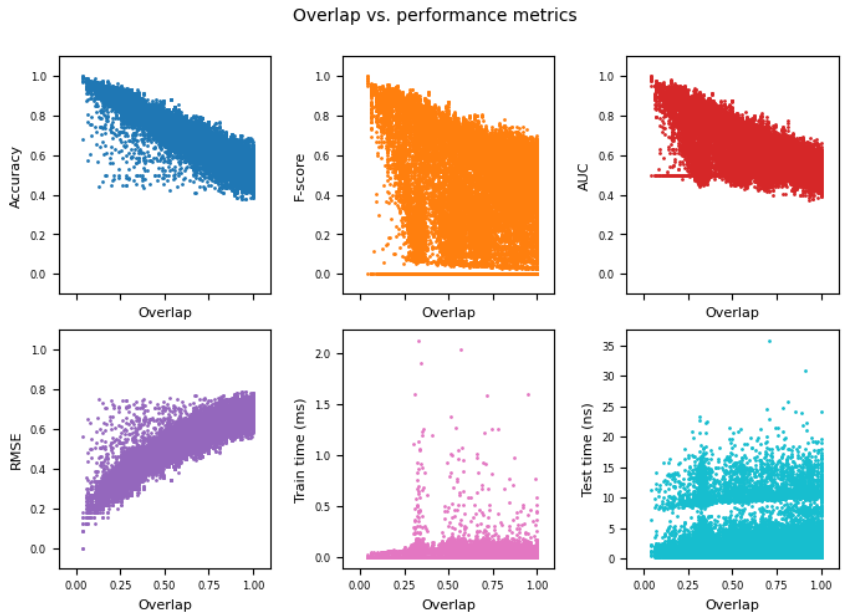


Figura 4.21: Relação de sobreposição calculada de classes e desempenho medido por todas as métricas de desempenho utilizadas.

A Figura 4.22 mostra a distribuição de valores calculados para sobreposição de classes. Devido à forma em que este valor é calculado – obtendo o resultado do algoritmo de R-value de OH (2011), multiplicando-o por 2, e limitando o resultado superiormente a 1 – espera-se um pequeno pico de ocorrências em 1, que agrupa todos os valores de R-value ≥ 0.5. Contudo, também percebe-se outro pico, ainda maior, entre 0.3 e 0.4. O motivo desse segundo pico não é claro; isso se torna um ponto de discussão para a seção 6.2.

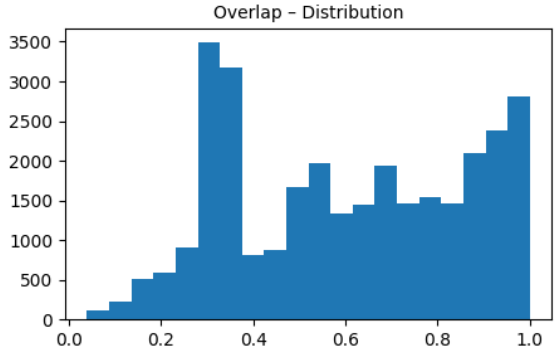


Figura 4.22: Distribuição de valores de sobreposição calculada para classes.

Capítulo 5

Dados reais

Como os experimentos iniciais não mostraram resultados muito distintos entre o desempenho dos classificadores, paralelamente às modificações descritas na seção 4.1, decidiu-se utilizar também dados reais para os testes, e não somente sintéticos. O pensamento é que dados reais podem ser mais desafiadores aos modelos, e podem ajudar a encontrar ocasiões de vantagens ou desvantagens de algoritmos individuais mais facilmente, de forma que não seria possível somente com dados sintéticos.

5.1 Revisão da literatura

Durante a busca na literatura pelo consenso sobre a otimização de parâmetros para diferentes classificadores, também considerou-se os conjuntos de dados reais usados pelos trabalhos encontrados. Ainda que nem todos os trabalhos explicitem os exatos dados usados, todos indicam, pelo menos de forma genérica, a fonte dos dados (Tabela 5.1).

Dos 19 artigos mais relevantes estudados, grande parte deles (11, ou 57.9%) faz uso de dados da UCI; em alguns casos (4, ou 21.1%), de forma exclusiva. Estas quantidades podem ser ainda maiores, pois outros dois (10.5%) utilizam dados providos pelo Weka,[1] que em certos casos também podem ser oriundos da UCI. Além disso, dois (10.5%) fazem uso de *datasets* obtidos pelo OpenML,[2] um agregador de dados aberto para uso livre por pesquisadores.

No total, são 14 (73.7%) trabalhos com dependência em agregadores de dados comumente usados. Em linha com as críticas feitas por JAPKOWICZ e SHAH (2011), é possível que isso crie "pontos cegos" e enviesamento dos resultados, pois algoritmos são sempre testados com os mesmos *datasets*, e raramente com problemas

[1]The Weka Workbench. *University of Waikato*. Disponível em: `https://ml.cms.waikato.ac.nz/weka`. Acesso em 12 jul.2024.

[2]OpenML. *The Open Machine Learning Foundation*. Disponível em: `https://openml.org/`. Acesso em 12 jul.2024.

reais inéditos.

Desta forma, houve um esforço consciente para o presente trabalho no sentido de diversificar a origem dos dados reais usados e não limitar-se a somente um repositório de dados.

Tabela 5.1: Dados reais usados por trabalhos relevantes encontrados na revisão de literatura

Trabalho	**Fonte dos dados**
CHAPELLE *et al.* (2002)	UCI
LAVESSON e DAVIDSSON (2006)	UCI
ABU ALFEILAT *et al.* (2019)	UCI
INYANG *et al.* (2023)	UCI
EL HINDI *et al.* (2020)	UCI, Weka
WICKRAMASINGHE e KALUTARAGE (2021)	UCI, Mendeley, Kaggle
CARUANA e NICULESCU-MIZIL (2004)	UCI, outros
BATISTA e SILVA (2009)	UCI, outros
SYARIF *et al.* (2016)	UCI, outros
XIA *et al.* (2017)	UCI, outros
OSHIRO *et al.* (2012)	UCI, dados médicos
PROBST e BOULESTEIX (2018)	OpenML
PROBST *et al.* (2019)	OpenML
EL HINDI *et al.* (2018)	Weka
ALONSO *et al.* (2017)	Dados médicos
HANCOCK e KHOSHGOFTAAR (2021)	Dados médicos
HOSSAIN *et al.* (2022)	Dados de gado
ANGHEL *et al.* (2019)	*Datasets* para ranking
ELGELDAWI *et al.* (2021)	Avaliações do Booking.com

5.2 Conjuntos de dados escolhidos

5.2.1 Higher Education Students Performance Evaluation

O primeiro *dataset* avaliado foi produzido por YILMAZ e SEKEROGLU (2020), e está disponível no repositório da UCI sob o número de identificação 856.[3] Os autores aplicaram um questionário de 30 perguntas a 101 estudantes de ensino superior e utilizaram quatro algoritmos – *Backpropagation Neural Network* (BNN), *RBF Neural Network* (RBFNN), *Decision Tree* (DT) e *Logistic Regression* (LogR) – para prever a nota de cada aluno, dentre oito opções, de acordo com a política de pontuação da universidade turca avaliada – AA, BA, BB, CB, CC, DC, DD e FF.

Quando utilizando todas as perguntas, os autores obtiveram, separadamente por curso do aluno, os desempenhos descritos na Tabela 5.2. O melhor desempenho foi

[3]Higher Education Students Performance Evaluation. *UC Irvine Machine Learning Repository*. Disponível em: `https://archive.ics.uci.edu/dataset/856/`. doi:10.24432/C51G82. Acesso em 17 jul.2024.

pelo modelo de RBFNN, seguido por BNN (exceto para o curso 3, cujo segundo melhor desempenho foi por DT).

Tabela 5.2: Acurácia reportada por YILMAZ e SEKEROGLU (2020) para todas as perguntas do questionário

	BNN	**RBFNN**	**DT**	**LogR**
Curso 1	65.00%	80.00%	42.00%	60.00%
Curso 2	76.90%	77.33%	75.00%	75.00%
Curso 3	33.33%	85.00%	50.00%	16.66%
Combinado	66.66%	70.00%	43.33%	53.33%

No entanto, algumas modificações são necessárias para o uso desse conjunto de dados em classificação binária. Primeiramente, as classes precisam ser reduzidas a apenas duas. Aqui, as notas AA, BA e BB foram consideradas "boas" e categorizadas como classe positiva, enquanto as outras foram consideradas "ruins", e colocadas na classe negativa. O objetivo dos classificadores, portanto, é corretamente prever se um aluno terá bom desempenho (classe positiva) ou não (classe negativa). O conjunto de dados tem, assim, 145 pontos, dos quais 98 (67.59%) são da classe negativa, e 47 (32.41%) da classe positiva. Finalmente, as colunas "Course ID" e "Student ID" foram removidas inteiramente, de forma a não enviesar os classificadores.

Novamente, devido ao tamanho diminuto do *dataset* (145 pontos), os classificadores foram testados separando os dados aleatoriamente em 75% para treino e 25% para teste, que traduziu-se em 37 pontos para teste e 108 para treino. A otimização de parâmetros foi, como em outras ocasiões no trabalho, realizada com Random Search com 25 iterações e validação cruzada de 5 folds.

A Tabela 5.3 agrega os desempenhos medidos pelos classificadores escolhidos pelo trabalho. Os três melhores desempenhos por cada métrica estão enfatizados em suas respectivas colunas, além do pior desempenho.

Tabela 5.3: Desempenho dos classificadores para o problema adaptado de notas de estudantes

Modelo	**Acurácia**	**F-score**	**AUC**	**RMSE**	**Tempo total (ms)**
kNN	**0.649**	**0.480**	**0.619**	**0.593**	**9.673**
Random Forest	0.595	0.118	0.449	0.637	471.876
Naive Bayes	**0.622**	**0.462**	**0.600**	**0.615**	**11.269**
Naive Bayes (PT)	0.595	**0.483**	**0.607**	0.637	1,102.291
GBDT	0.595	0.348	0.528	0.637	227.682
GBDT (PT)	0.595	0.286	0.502	0.637	**34,571.668**
SVM pré-comp.	0.568	0.333	0.509	0.658	17.388
SVM RBF	0.568	0.200	**0.456**	0.658	33.956
SVM RBF (PT)	**0.514**	0.308	0.470	**0.698**	1,308.860
SVM Pol.	0.595	0.348	0.528	0.637	**14.551**
SVM Pol. (PT)	**0.703**	**0.000**	0.500	**0.545**	1,235.312

É importante ressaltar que os desempenhos encontrados aqui e por YILMAZ e SEKEROGLU (2020) não são estritamente comparáveis; o problema foi modificado para classificação binária, com modelos e objetivos distintos. Assim, os desempenhos dos autores são apresentados acima apenas como referência.

De forma geral, o classificador com consistentemente melhor desempenho neste *dataset* foi o kNN. Ele possui o melhor ou o segundo melhor desempenho em todas as métricas, característica única a ele. Ainda assim, seu desempenho não é nada extraordinário: com 65% de acurácia e 48% de F-score, ele pode ser considerado até um classificador *fraco*, e só está no topo pois os outros modelos são ainda mais inadequados ao problema. Sua matriz de confusão está presente na Tabela 5.4.

Tabela 5.4: Matriz de confusão do kNN para o problema adaptado de notas de estudantes

	Predição Positiva	**Predição Negativa**	**Total:**
Positivo	**6** (True Positive)	**5** (False Negative)	11 29.73%
Negativo	**8** (False Positive)	**18** (True Negative)	26 70.27%
Total:	14 37.84%	23 62.16%	37 100%

Destaca-se, também, o desempenho do SVM com *kernel* polinomial e otimização de parâmetros, que obteve $> 70\%$ de acurácia, mas F-score de 0. Ao analisar sua matriz de confusão (Tabela 5.5), o motivo fica aparente: o conjunto de dados é desbalanceado, com mais pontos na classe negativa do que na positiva. Assim, um modelo que classifica sempre negativamente, ao ser testado, alcança acurácias altas, mesmo que não tenha um bom poder de predição.

Tabela 5.5: Matriz de confusão do SVM polinomial otimizado para o problema adaptado de notas de estudantes

	Predição Positiva	**Predição Negativa**	**Total:**
Positivo	**0** (True Positive)	**11** (False Negative)	11 29.73%
Negativo	**0** (False Positive)	**26** (True Negative)	26 70.27%
Total:	0 0%	37 100%	37 100%

5.2.2 McDonald's Stock Price

O conjunto de dados seguinte vem da plataforma Kaggle, disponibilizado por Clemence Travers.[4] O *dataset* contém valores de diversos índices de ações semanais por cinco anos, entre 2019 e 2024. Ele tem como objetivo prever se as ações do McDonald's subiram (classe positiva) ou caíram (classe negativa) ao final da semana. Dentre as informações disponíveis, estão: as variações das ações da Coca-Cola e do Starbucks; o preço de óleo e açúcar; e a presença da guerra na Ucrânia.

A distribuição entre as classes é equilibrada: de um total de 1257 pontos, são 601 (47.81%) na classe negativa e 656 (52.19%) na classe positiva. Novamente, por ser um *dataset* relativamente pequeno, os pontos foram separados em 75%/25% para treino e teste. A otimização de parâmetros, para os modelos que a utilizaram, manteve-se via Random Search com 25 iterações e validação cruzada de 5 folds.

A Tabela 5.6 detalha os desempenhos de cada classificador para este conjunto de dados em cada uma das métricas selecionadas. Estão enfatizados os três melhores desempenhos em cada métrica, além do pior desempenho, em suas respectivas colunas.

Tabela 5.6: Desempenho dos classificadores para o problema de ações do McDonald's

Modelo	**Acurácia**	**F-score**	**AUC**	**RMSE**	**Tempo total (ms)**
Random Forest	0.595	0.118	0.449	0.637	471.876
kNN	0.502	0.560	0.493	0.706	**20.514**
Random Forest	**0.486**	**0.526**	0.484	**0.717**	1,034.447
Naive Bayes	0.537	**0.674**	**0.483**	0.681	**12.420**
Naive Bayes (PT)	**0.546**	**0.686**	0.489	**0.674**	1,128.216
GBDT	0.502	0.558	0.494	0.706	522.339
GBDT (PT)	0.502	0.548	0.498	0.706	**169,495.752**
SVM pré-comp.	0.514	0.576	**0.504**	0.697	**57.784**
SVM RBF	0.508	0.573	0.496	0.702	228.262
SVM RBF (PT)	**0.546**	0.638	**0.519**	**0.674**	14,197.335
SVM Pol.	0.530	0.651	0.488	0.686	125.195
SVM Pol. (PT)	**0.578**	**0.732**	**0.500**	**0.650**	11,880.301

Novamente, o SVM com *kernel* polinomial e otimização de parâmetros surpreende, dessa vez posicionando-se com o melhor por múltiplas métricas, e sem um F-score nulo. Contudo, por trás dos panos, ele continua sofrendo do mesmo problema de antes: ao verificar-se sua matriz de confusão (Tabela 5.7), é imediatamente evidente que o algoritmo só está classificando todos os pontos como positivos. Assim, ele não pode ser considerado um bom preditor, mesmo que as métricas de desempenho usadas digam o contrário. Uma discussão sobre essa situação está presente na seção 6.1.

[4] McDonald's Stock Price. *Kaggle*. Disponível em: `https://kaggle.com/datasets/clemencetravers/predict-mc-donalds-stock-price`. Acesso em: 17 jul.2024.

Tabela 5.7: Matriz de confusão do SVM polinomial otimizado para o problema de ações do McDonald's

	Predição Positiva	**Predição Negativa**	**Total:**
Positivo	**182** (True Positive)	**0** (False Negative)	182 57.78%
Negativo	**133** (False Positive)	**0** (True Negative)	133 42.22%
Total:	315 100%	0 0%	315 100%

O segundo melhor desempenho como declarado pelas métricas usadas é pelo SVM com *kernel* gaussiano e otimização de parâmetros. Comparativamente, ele é fantástico – até faz predições negativas, ainda que mais falsas do que verdadeiras. A Tabela 5.8 mostra a matriz de confusão para o classificador.

Tabela 5.8: Matriz de confusão do SVM RBF otimizado para o problema de ações do McDonald's

	Predição Positiva	**Predição Negativa**	**Total:**
Positivo	**126** (True Positive)	**56** (False Negative)	182 57.78%
Negativo	**87** (False Positive)	**46** (True Negative)	133 42.22%
Total:	213 67.62%	102 32.38%	315 100%

5.2.3 NBA Rookies' Career

Este é um *dataset* disponibilizado pelo website data.world, adaptado de dados de jogadores de basquete dos Estados Unidos agregados por Gabe Salzer.[5] Ele é preparado com o objetivo de prever a longevidade da carreira de diversos jogadores; especificamente, se a carreira dura menos de cinco anos (classe negativa) ou mais (classe positiva).

Os dados são, em geral, estatísticas de jogo – como participação em jogos, pontos por jogo, acertos de cestas de três pontos, entre outros. Eles totalizam 19 *features*, além do nome do jogador (descartado para os experimentos) e o resultado, negativo se a carreira durou menos que cinco anos, e positivo se durou cinco ou mais. São 1316 pontos distintos, dos quais 499 (37.9%) são da classe negativa, e 817 (62.1%) são da classe positiva.

[5]Classification Exercise: Predict 5-Year Career Longevity for NBA Rookies. *data.world*. Disponível em: `https://data.world/exercises/logistic-regression-exercise-1`. Acesso em 29 ago.2024.

Mais uma vez, por se tratar de um conjunto com relativamente pouco dados, os pontos foram separados em 75%/25% para treino e teste. A otimização de parâmetros foi realizada via Random Search com 25 iterações e validação cruzada de 5 folds. A Tabela 5.9 a seguir contém os desempenhos por classificador, medidos por cada métrica, para este conjunto de dados.

Tabela 5.9: Desempenho dos classificadores para o problema de carreira no NBA

Modelo	**Acurácia**	**F-score**	**AUC**	**RMSE**	**Tempo total (ms)**
kNN	0.705	0.763	0.685	0.543	**82.326**
Random Forest	0.693	0.768	0.654	0.554	895.409
Naive Bayes	**0.617**	**0.609**	0.654	**0.619**	**10.369**
Naive Bayes (PT)	**0.733**	0.783	**0.716**	**0.517**	1,978.111
GBDT	0.666	0.737	**0.637**	0.578	533.254
GBDT (PT)	0.705	0.776	0.668	0.543	**251,954.441**
SVM pré-comp.	**0.723**	**0.785**	**0.694**	**0.526**	3,200.983
SVM RBF	0.714	**0.783**	0.677	0.535	196.644
SVM RBF (PT)	0.720	**0.785**	0.687	0.529	11,848.123
SVM Pol.	**0.726**	0.782	**0.705**	**0.523**	**79.008**
SVM Pol. (PT)	0.711	0.771	0.687	0.537	7,764.791

Os resultados encontrados são muito oportunos e formam um ótimo argumento em prol da otimização de parâmetros. Nos testes, o classificador NB não otimizado obteve o pior desempenho em quase todas as métricas – sendo bem colocado somente em termos de tempo de execução, como o algoritmo mais veloz. Entretanto, ao introduzir a otimização de parâmetros, o NB se torna o melhor algoritmo em todas as métricas exceto tempo e F-score, ficando com o 4º melhor posicionamento neste último. Abaixo, Tabela 5.10 e Tabela 5.11 mostram as matrizes de confusão para NB e NB otimizado, respectivamente.

Tabela 5.10: Matriz de confusão do NB para o problema de NBA rookies

	Predição Positiva	**Predição Negativa**	**Total:**
Positivo	**98** (True Positive)	**103** (False Negative)	201 61.09%
Negativo	**23** (False Positive)	**105** (True Negative)	128 38.91%
Total:	121 36.78%	208 63.22%	329 100%

O único parâmetro de NB, *var_ smoothing*, possui valor padrão de implementação de 10^{-9}; após a otimização, ele se torna aproximadamente 0.35, a algumas ordens de grandeza de diferença. Assim, é evidente que o melhor desempenho não seria alcançado sem a otimização do parâmetro.

Outro algoritmo com boa *performance* é o SVM com *kernel* polinomial. Curiosamente, ele traz o argumento inverso, contra otimização de parâmetros: sua versão

Tabela 5.11: Matriz de confusão do NB otimizado para o problema de NBA rookies

	Predição Positiva	**Predição Negativa**	**Total:**
Positivo	**159** (True Positive)	**42** (False Negative)	201 61.09%
Negativo	**46** (False Positive)	**82** (True Negative)	128 38.91%
Total:	205 62.31%	124 37.69%	329 100%

out-of-the-box possui desempenho superior à versão com parâmetros otimizados – possivelmente devido ao não-determinismo do Random Search na busca de parâmetros. Vale notar que, diferentemente de outros testes com dados reais, aqui o SVM polinomial otimizado não classificou todos os pontos em uma única classe.[6]

Assim, na falta de uma busca maior e mais demorada por parâmetros, em certas ocasiões a implementação de *parameter tuning* pode ser em detrimento do desempenho.

[6] A matriz de confusão do SVM polinomial foi (TP: 161, FN: 40; FP: 50, TN: 78); a de sua versão otimizada, (TP: 160, FN: 41; FP: 54, TN: 74). Muito próximos, mas a otimização foi em detrimento do desempenho.

Capítulo 6

Discussão

Nos experimentos iniciais, percebeu-se, acima de tudo, a grande similaridade entre os algoritmos – e isso foi um tema recorrente ao longo do trabalho. Porém, há sempre pequenas diferenças que podem ser exploradas, e estas trouxeram uma tênue divisão entre os classificadores.

No primeiro grupo estão kNN, RF e GBDT, cujos resultados indicaram uma maior resistência ao desequilíbrio de dados e ao ruído. No segundo grupo, os SVMs e NB, mais "fracos" quando lidando com ruído e desequilíbrio.

Estes dois grupos, contudo, não se mantiveram em termos de tempo de treino; ao invés de disso, formaram-se três divisões: algoritmos lentos (RF e GBDT), meio-termo (SVMs), e rápidos (kNN e NB). Os tempos do SVM de *kernel* pré-computado, especificamente, tiveram uma variância muito alta, abrangendo as duas divisões mais lentas.

Por sua vez, nos experimentos revisados, manteve-se a perspectiva de similaridade no desempenho dos algoritmos – de forma geral, todos são igualmente adequados para a tarefa de classificação, pelo menos no que tange aos conjuntos de dados utilizados.

Contudo, há algumas diferenças na execução dos algoritmos. Devido à imprescIência na forma de não incluir o cálculo da matriz de confusão ou métrica similar, não é possível dizer com absoluta certeza mas, ao que tudo indica, alguns classificadores são mais propensos a classificar todos os pontos em uma classe só, independentemente de otimização de parâmetros.

Novamente, encontra-se uma separação dos classificadores em dois grupos: kNN, RF e GBDT parecem mais resistentes a esse tipo de erro, classificando de forma mais equilibrada; os SVMs e o NB, por outro lado, possuem muito mais ocorrências de F-score = 0. No caso do SVM de *kernel* polinomial com parâmetros otimizados, são dez vezes mais instâncias quando comparando-o com os algoritmos do primeiro grupo. Devido ao funcionamento do F-score, presume-se que esse erro indica a classificação de todos os pontos na classe negativa, como evidenciado na seção 5.2.1.

Mais uma vez, em termos de tempo, agora levando em consideração treino e teste, NB é o algoritmo mais rápido, seguido de perto por kNN. A etapa de treino é, de modo geral, mais demorada que a de teste; assim, algoritmos com pouco ou nenhum treino – como kNN e NB – tendem a ser mais rápidos no tempo total.

O classificador com tempo mais imprevisível – isto é, com maior variância – ainda é SVM com *kernel* pré-computado. RF é consideravelmente lento, com duração comparável à de classificadores cujo tempo inclui otimização de parâmetros, mesmo que ele próprio não possua essa etapa. Por fim, GBDT com parâmetros otimizados foi o algoritmo mais lento de todos, com mediana aproximadamente quatro ordens de grandeza maior que as dos classificadores mais rápidos, kNN e NB.

Ainda no que tange ao tempo, quando se tratando de problemas individuais, é importante pontuar que a correlação entre tempo e desempenho parece não ser tão relevante. Problemas mais "fáceis" – por exemplo, classes linearmente separáveis – são mais rápidos de resolver e, paralelamente, também alcançam melhores desempenhos. Mas, crucialmente, isso não significa que um algoritmo mais rápido é consequentemente melhor que um mais lento.

E, por fim, uma limitação na medida de tempo vem do fato de que, devido à divisão de 75%/25% dos dados entre treino e teste, houve menos pontos sendo testados – e, portanto, ainda que ambos fossem equivalentes, o tempo de teste medido seria $\frac{1}{3}$ do tempo de treino. Assim, uma análise talvez ainda mais interessante do que a realizada seria para *datasets* separados em 50%/50% para treino e teste, onde poderia-se desenhar uma comparação mais direta entre os tempos de cada modelo.

Outro ponto relevante na escolha por algoritmos são as pressuposições que cada classificador faz. Por exemplo, o algoritmo de NB presume dimensões independentes nos dados; no caso dos dados artificiais utilizados, isso é verdade – os pontos são gerados aleatoriamente via distribuição Normal, sem dependência entre *features* – então seu desempenho será melhor nesses dados "de brinde". Consequentemente, em outros conjuntos de dados, o algoritmo talvez demonstre um pior desempenho.

Para além disso, como explorado anteriormente, é essencial que a otimização de parâmetros, quando realizada, seja feita de forma inteligente, com foco em um conjunto de dados ou problema específico. Caso contrário, o desempenho do algoritmo pode não somente manter-se igual em comparação com seu uso com parâmetros padrão, mas até mesmo decair.

Essa foi a situação encontrada nos resultados dos experimentos – não houve uma instância de melhora significativa de desempenho ao introduzir a otimização de parâmetros, mas sim o oposto: muitas ocorrências de piora do desempenho com a otimização. Essa e outras questões similares são exploradas na seção 6.3.

6.1 Skew

No decorrer dos experimentos, evidenciou-se que as métricas de desempenho escolhidas, mesmo após serem modificadas, ainda possuem algumas limitações. Em especial, deparando-se com dados desbalanceados, há ocasiões em que um classificador pode ser avaliado positivamente, mesmo não tendo aprendido a classificar dados adequadamente.

Por exemplo, na seção 5.2.2, o algoritmo de SVM com *kernel* polinomial e otimização de parâmetros alcançou o melhor desempenho em quase todas as métricas, comparado aos outros classificadores. Ainda assim, ao verificar a matriz de confusão, o algoritmo fez 100% de suas predições na classe positiva, e nenhuma na classe negativa. Esse é o clássico exemplo do problema por trás do uso da acurácia como métrica de desempenho, e uma situação que esperava-se que a introdução de outras métricas amenizaria.

Adicionalmente, há a consideração de que o classificador teve seus parâmetros otimizados. Mas, crucialmente, a métrica usada pelo Random Search para otimizar os parâmetros é acurácia![1] Assim, as falhas são propagadas – otimiza-se para uma métrica errônea e obtém-se um ótimo desempenho, medido pela mesma métrica.

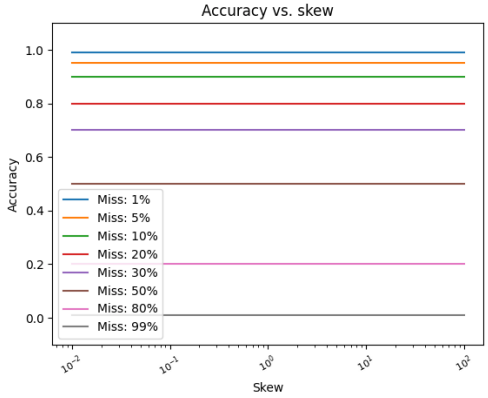
JENI *et al.* (2013), avaliando o impacto de dados desbalanceados, analisam o efeito de *skew* ou assimetria dos dados. Os autores definem *skew* como a razão entre pontos na classe negativa e pontos na classe positiva, e desenham a variação deste valor contra o desempenho de classificadores simulados medidos por diferentes métricas. Esses gráficos foram recriados utilizando o código do Apêndice C e são mostrados a seguir, com métricas adicionais.

Percebe-se que o *skew*, ou desequilíbrio dos dados, não afeta o desempenho medido por acurácia, AUC e RMSE (Figura 6.1(a,c,d)). Assim, cria-se um ponto cego: um algoritmo, como o SVM polinomial PT descrito acima, pode sempre classificar pontos em uma mesma classe e, a depender da distribuição dos dados, obter um desempenho satisfatório.

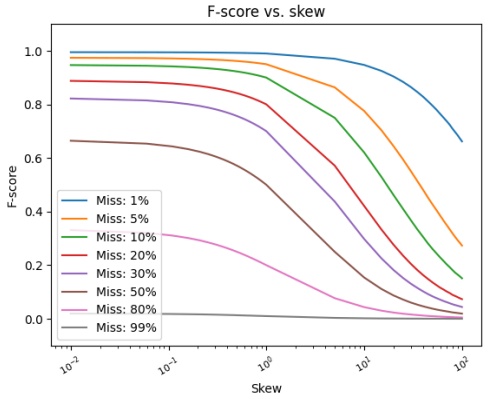
No caso de F-score (Figura 6.1(b)), há alguma resistência a *skew*; mas, é preciso enfatizar, somente a *skew* em uma direção. Isso é evidenciado na seção 5.2.1, na qual o F-score para o SVM polinomial PT foi 0%, pois a predição foi inteiramente na classe negativa; em contraste com o F-score de 73% com predição inteiramente na classe positiva na seção 5.2.2.

Verifica-se o mesmo comportamento da acurácia nas métricas utilizadas para os primeiros experimentos, mas posteriormente removidas (Figura 6.1(e,f)), o que sustenta a decisão por sua remoção, pois elas não solucionam este ponto cego.

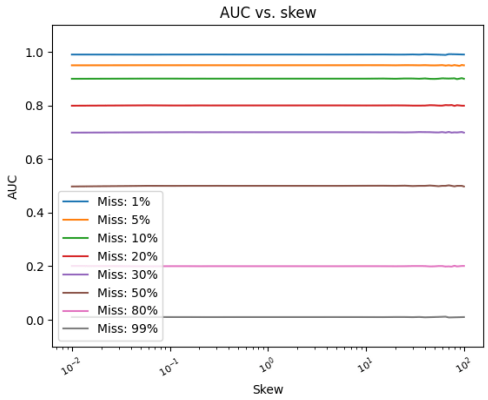
[1] Por padrão, 'sklearn.model_selection.RandomizedSearchCV' usa função de *score* do modelo sendo otimizado; neste caso, é 'sklearn.svm.SVC.score', que usa a média de acurácias.



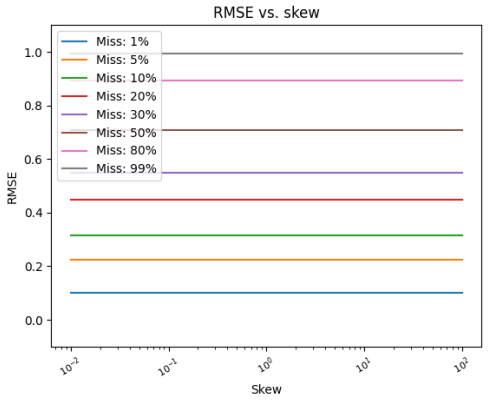
(a) Desempenho medido por acurácia.



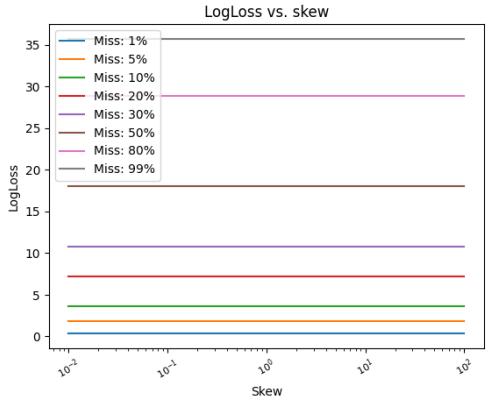
(b) Desempenho medido por F-score.



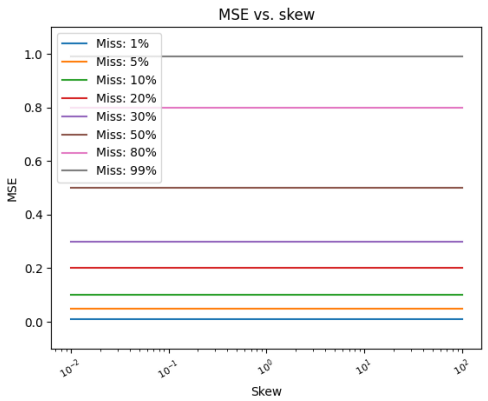
(c) Desempenho medido por AUC.



(d) Desempenho medido por RMSE.



(e) Desempenho medido por LogLoss.

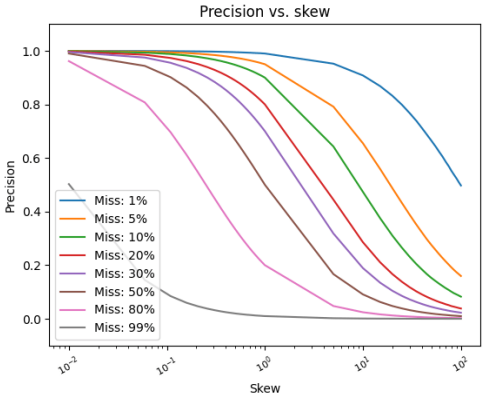


(f) Desempenho medido por MSE.

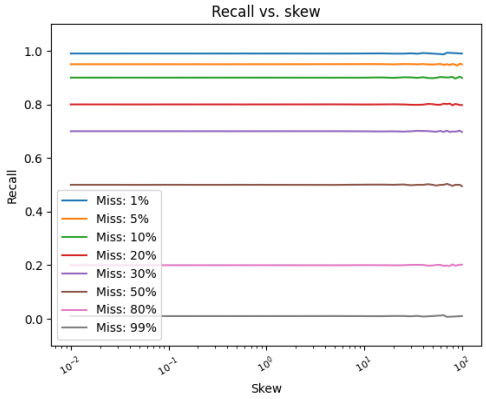
Figura 6.1: Gráficos *skew* vs. métrica de desempenho, para simulações de acertos por um classificador hipotético. Cada linha representa um percentual de erro de predição. O eixo horizontal é logarítmico.

Finalmente, na Figura 6.2, evidencia-se o comportamento das métricas de Precision e Recall, que são a base para o cálculo do F-score. Percebe-se que a sensibilidade ao *skew* se dá exclusivamente por conta da métrica de Precision, e não Recall.

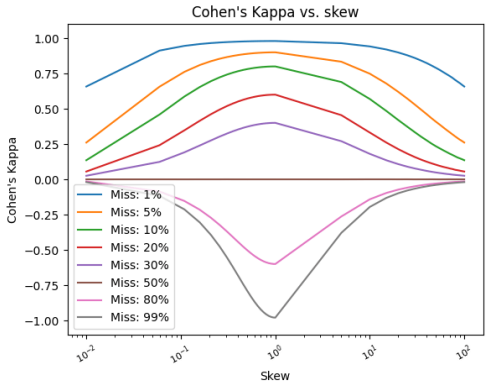
A Figura 6.2(c) também mostra a relação com a métrica de *Kappa* de COHEN (1960), uma das avaliadas por JENI *et al.* (2013). Essa métrica reage tanto ao *skew*



(a) Desempenho medido por Precision.



(b) Desempenho medido por Recall.



(c) Desempenho medido por Kappa de Cohen.

Figura 6.2: Gráficos *skew* vs. métrica de desempenho, para simulações de acertos por um classificador hipotético. Cada linha representa um percentual de erro de predição. O eixo horizontal é logarítmico.

positivo quanto negativo e, portanto, seu uso pode cobrir os pontos cegos de outras métricas.

Finalmente, outra métrica de mais simples implementação e que também ajuda a evidenciar baixa habilidade de predição em dados com *skew* alto é a própria matriz de confusão.

6.2 Sobreposição

Outra métrica calculada que se mostrou inadequada – ou, mais corretamente, insuficiente para os experimentos – foi a métrica de sobreposição de classes R-value de OH (2011), que, similarmente ao caso das métricas de desempenho, possuiu ponto cego para algumas situações de desequilíbrio de dados.

Por exemplo, a Figura 6.3 mostra uma situação de desequilíbrio "extremo": uma classe possui 1000 pontos, enquanto a outra possui somente 10. Além dessa dis-

paridade, todos os outros parâmetros são idênticos – e, portanto, a sobreposição calculada deveria atingir o mais próximo possível de 100%. Contudo, o R-value calculado é de apenas 0.01.

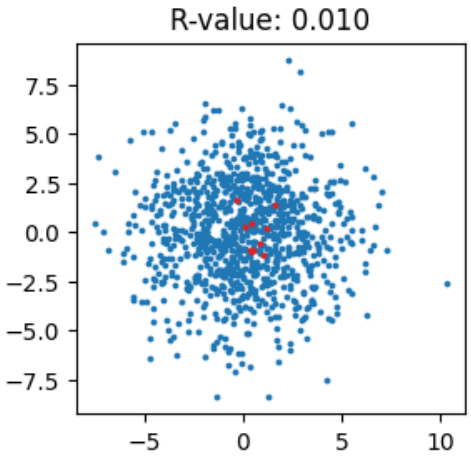


Figura 6.3: Cálculo de R-value para duas classes geradas com parâmetros idênticos – valor esperado de 0.005 e variância de 0.05 – com exceção do número de pontos (1000 vs. 10).

Além disso, experimentalmente, a métrica de R-value só retornou 100% de sobreposição para conjuntos de dados idênticos. Assim, com os dados gerados, mesmo com classes equilibradas e com sobreposições eclípticas, o R-value calculado raramente ultrapassou 60%, o que instigou a mudança para uso de R-value $\times$ 2.

É possível que, através do refinamento dos parâmetros da medida, K e θ, seja possível obter uma melhor eficácia na aferição de sobreposição. Senão, uma métrica diferente, que leve em consideração as insuficiências apontadas, seria mais adequada.

6.3 Questões não resolvidas

Após a análise dos resultados, algumas questões permaneceram inexplicadas. Devido ao limite de tempo e escopo do trabalho, estas se tornam possibilidades para estudos posteriores e são mencionadas na seção 7.1.

A primeira situação inusitada foi percebida ao verificar a distribuição de valores calculados para a sobreposição de classes, na seção 4.3.7. A métrica é calculada a partir do R-value, dobrado e limitado a 1. Assim, esperaria-se uma maior quantidade de 1's, contando instâncias de R-value ≥ 0.5 que, após serem duplicadas, foram limitadas a 1. Entretanto, o maior pico é, na realidade, em torno de 0.3.

O motivo para esse pico não é claro. É possível que, devido a relações complexas nas interações entre o gerador de dados e o algoritmo de cálculo de sobreposição, a distribuição do valor não seja linear, e 0.3 representa a mediana. Também é possível

que a geração de dados seja falha e tenha uma tendência a produzir dados parecidos, com uma mesma sobreposição

Outra situação curiosa surge na diferença percebida no desempenho dos algoritmos antes e depois da otimização, na seção 4.3.6. Em teoria, devido ao funcionamento da otimização de parâmetros, espera-se que um modelo não-otimizado seja pior, ou pelo menos tão bom quanto um modelo otimizado. Afinal, se a escolha de parâmetros padrão era ideal ou ótima, a otimização deveria selecionar os mesmos parâmetros.

Mas isso não foi o ocorrido e, de fato, os modelos otimizados tiveram desempenhos consistentemente piores que os não otimizados. Novamente, o motivo não é perfeitamente claro. Devido ao uso de Random Search, os parâmetros são escolhidos de forma aleatória; assim, não seria impossível que a combinação de valores dos parâmetros padrão não tenha sido selecionada e, por acaso, seria a melhor opção. Além disso, como mencionado anteriormente, a métrica de desempenho usada para refinar o Random Search é acurácia, então, os algoritmos são otimizados para uma melhor acurácia, o que não necessariamente significa um melhor desempenho em geral.

Finalmente, em uma análise de ocorrências de AUC ≤ 0.5 na seção 4.3.4, percebeu-se que alguns algoritmos – kNN, GBDT (com e sem otimização) e RF – possuem relativamente poucas instâncias de AUC $= 0.5$; em contrapartida, os SVMs e NB, independentemente de otimização, obtém muitas, mas menos AUC < 0.5 que o primeiro grupo. O motivo por trás dessa separação de erros é desconhecido.

Um desempenho medido por AUC de 0.5 costuma indicar um algoritmo que não aprendeu nada; ele está equivalente a cara e coroa de uma moeda. Porém, um desempenho abaixo de 0.5 significa que o algoritmo conseguiu, sim, aprender algo sobre os dados, mas ele está classificando erroneamente – isto é, é possível obter informações suficientes para distinguir, pelo menos em parte, as classes, mas o classificador está usando essas informações incorretamente. Por que alguns algoritmos são mais resistentes a um erro, mas menos ao outro, não é claro.

Capítulo 7

Conclusão

Neste trabalho, buscou-se encontrar distinções entre o desempenho de diversos algoritmos de classificação binária supervisionada, com o objetivo de orientar novos trabalhos sobre o comportamento de classificadores a depender das características dos dados. Os classificadores testados, implementados pela biblioteca scikit-learn de Python, foram: *Stochastic Gradient Boosted Decision Trees* (GBDT), *Gaussian Naive Bayes* (NB), *Random Forest* (RF), *K-Nearest Neighbors* (kNN) e *SVM* com 3 *kernels* distintos (pré-computado, gaussiano ou RBF, e polinomial).

Para evitar vieses e possível *overfitting*, utilizou-se tanto dados gerados artificialmente quanto dados reais, e todos os experimentos foram realizados em máquinas virtuais padronizadas, em nuvem. Foram incluídos classificadores tanto com seus parâmetros padrão de implementação da biblioteca, quanto com uma etapa de otimização de parâmetros via Random Search.

O desempenho dos classificadores foi medido a partir de nove métricas distintas no total. Em experimentos iniciais, foram mensurados pelas seguintes seis: acurácia ou *Overall Success Rate*, F-score ou F-measure, LogLoss, *Area Under ROC Curve* (AUC), *Mean Squared Error* (MSE) e tempo de treino; e, em experimentos revisados, pelas seguintes sete: acurácia, F-score, *Root Mean Squared Error* (RMSE), AUC e três métricas de tempo (treino, teste e total).

Nos experimentos, percebeu-se uma grande similaridade no desempenho dos classificadores. Assim, primeiramente, é importante explicitar que todos são algoritmos adequados, e nenhum teve desempenho significativamente inferior ou superior aos outros, exceto em termos de tempo de execução.

Isso é esperado e, pelo menos em parte, explicado pelo viés de sobrevivência: algoritmos com desempenhos significativamente abaixo dos outros não se tornariam populares e, portanto, receberiam menos atenção, menos estudos, menos implementações. Como a escolha de classificadores foi realizada a partir de endossos de estudos anteriores, não é surpresa que nenhum deles destaque-se como sendo o pior.

Isso dito, os classificadores podem, de modo geral, ser separados em dois grupos,

com diferentes comportamentos em face a condições adversas. No primeiro grupo, estão kNN, RF e GBDT; e, no segundo, os SVMs e NB. Essa separação se mostrou presente em várias situações.

Primeiramente, o primeiro grupo possui maior resistência à introdução de ruído nos dados, ou a classes desequilibradas – isto é, sem um número igual de pontos em cada classe. Em termos simples, kNN, RF e GBDT tenderam a um melhor desempenho conforme ruído ou desequilíbrio foi introduzido, enquanto SVMs e NB tiveram uma piora mais rápida.

Além disso, o segundo grupo, de SVMs e NB, teve, em média, quantidade quase oito vezes maior de ocorrências de F-score = 0 e quase 45 vezes maior de ocorrências de AUC = 0.5, quando comparado à média do primeiro grupo, de kNN, RF e GBDT. Isso indica que o segundo grupo teve maior dificuldade em obter informações a partir dos dados quando comparado ao primeiro grupo.

Em especial, o SVM com *kernel* polinomial e otimização de parâmetros foi o maior ofensor, em ambos os casos se posicionando com a maior quantidade de ocorrências desses erros. Como também demonstrado nos testes com dados reais, esse classificador parece ter uma maior tendência de classificar todos os pontos em uma única classe, seja positiva ou negativa.

A maior diferença de desempenho dos algoritmos, contudo, é no quesito tempo. A divisão em dois grupos, aqui, não mais se aplica; mas, sim, uma outra separação, em três: algoritmos lentos (RF e GBDT), meio-termo (SVMs), e rápidos (kNN e NB). O algoritmo mais lento, GBDT com otimização, obteve tempos quatro ou cinco ordens de grandeza mais lentos que os melhores modelos, kNN e NB. Por sua vez, os tempos do SVM de *kernel* pré-computado tiveram uma variância muito alta, abrangendo as duas divisões mais lentas.

Finalmente, é essencial que a otimização de parâmetros, quando realizada, seja feita de forma inteligente, com foco em um conjunto de dados ou problema específico. Caso contrário, o desempenho do algoritmo pode não somente manter-se igual em comparação com seu uso com parâmetros padrão, mas até mesmo decair.

Assim, no escopo dos dados testados, não havendo outros fatores a serem considerados para além dos mencionados anteriormente, recomenda-se o uso do kNN como classificador binário supervisionado, por sua relativa maior resistência a ruído e desequilíbrio de dados; por seu curto tempo de execução; e por dispensar a necessidade de otimização de parâmetros.

7.1 Trabalhos futuros

Ainda há uma ampla gama de questões a serem respondidas e experimentos a serem executados no tópico abordado por este trabalho.

Por exemplo, tratando-se dos dados artificiais, um parâmetro que permaneceu inalterado é N, o número de pontos gerados para classificação. Pode-se derivar perguntas interessantes que motivam essa oportunidade: diferentes valores para N possuem diferentes impactos no desempenho dos algoritmos? Ou, até, mais relevante: algoritmos reagem de forma diferente a mudanças de valores para N? Se sim, isso permitiria a escolha de classificador com base no número de pontos (e.g. "para $N < 5000$, kNN é recomendado").

Além disso, poucos experimentos foram realizados com a variação do número de dimensões dos pontos. Esta alteração pode ter um impacto mais profundo do que o esperado, pois é possível que nem todos os algoritmos mantenham bom desempenho em problemas de muitas dimensões.

Outro ponto a ser considerado é o número de classes. Aqui, se tratando de classificadores binários, os experimentos foram restritos a duas classes – mas os resultados podem ser diferentes em situações multiclasse. Por exemplo, o algoritmo recomendado de kNN, segundo INYANG *et al.* (2023), de fato desempenha melhor em problemas de poucas classes. Portanto, em situações de múltiplas classes, talvez outro seja mais adequado.

Finalmente, a possibilidade de gerar dados seguindo outras distribuições – como, por exemplo, multimodal – também tem potencial de ser uma melhora significativa ao gerador de dados, pois expõe os classificadores a dados mais diversos e potencialmente mais complexos. Essa mudança poderia, ainda, ser acoplada à uma outra forma de avaliar os modelos: *feature importance*. Isto é, a determinação de quanto cada dimensão dos dados está influenciando a decisão dos modelos. Ao fazer uso de diferentes distribuições, espera-se ver uma não-uniformidade na importância de cada dimensão.

Em termos gerais, as métricas de desempenho escolhidas, mesmo após a revisão dos experimentos, ainda não foram ideais. Como sugerido na seção 6.1, a inclusão de uma métrica que reaja a ambas as formas de *skew* – positiva e negativa – é imprescindível em trabalhos futuros. Uma sugerida por JENI *et al.* (2013) é o *Kappa* de COHEN (1960); mas uma análise da matriz de confusão também poderia ser usada com essa finalidade.

Além disso, o cálculo de sobreposição de classes também tem potencial para melhora, como descrito na seção 6.2. É necessário que o valor leve em consideração o desequilíbrio de classes e seja mais representativo da sobreposição das distribuições, e não somente dos pontos individuais.

Adicionalmente, neste trabalho experimentou-se somente com implementações dos algoritmos em Python e, mais especificamente, da biblioteca scikit-learn. Ainda que o funcionamento dos algoritmos seja, em princípio, sempre o mesmo, diferentes implementações possuem diferentes otimizações que, no melhor dos casos, afetariam

pelo menos as métricas de tempo; mas potencialmente outras métricas, também.

Por fim, em termos de otimização de parâmetros, é necessário fazê-la "multiobjetivo" – isto é, otimizar para múltiplas métricas e não somente à acurácia, como ocorreu inopinadamente neste trabalho. Ademais, existem métodos promissores inexplorados, como *Bayesian Optimization* (NGUYEN, 2019) ou otimização de ponto de corte a partir da curva ROC (UNAL, 2017), cujo uso pode ser interessante.

Referências Bibliográficas

BLANK, G., REISDORF, B. C. "The Participatory Web", *Information, Communication & Society*, v. 15, n. 4, pp. 537–554, Mar. 2012. doi: 10.1080/1369118X.2012.665935.

MCGRADY, R., ZHENG, K., CURRAN, R., et al. "Dialing for Videos: A Random Sample of YouTube", *Journal of Quantitative Description: Digital Media*, v. 3, Dez. 2023. ISSN: 2673-8813. doi: 10.51685/jqd.2023.022. Disponível em: <https://journalqd.org/article/view/4066>.

ABDALLAH, A., MAAROF, M. A., ZAINAL, A. "Fraud detection system: A survey", *Journal of Network and Computer Applications*, v. 68, pp. 90–113, 2016. ISSN: 1084-8045. doi: https://doi.org/10.1016/j.jnca.2016.04.007. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1084804516300571>.

KARIM, A., AZAM, S., SHANMUGAM, B., et al. "A Comprehensive Survey for Intelligent Spam Email Detection", *IEEE Access*, v. 7, pp. 168261–168295, 2019. doi: 10.1109/ACCESS.2019.2954791.

SAIDANI, N., ADI, K., ALLILI, M. S. "A semantic-based classification approach for an enhanced spam detection", *Computers & Security*, v. 94, pp. 101716, 2020. ISSN: 0167-4048. doi: https://doi.org/10.1016/j.cose.2020.101716. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167404820300043>.

GUTTA, S., WECHSLER, H. "Face recognition using hybrid classifier systems". In: *Proceedings of International Conference on Neural Networks (ICNN'96)*, v. 2, pp. 1017–1022 vol.2, 1996. doi: 10.1109/ICNN.1996.549037.

HORIGUCHI, S., AMANO, S., OGAWA, M., et al. "Personalized Classifier for Food Image Recognition", *IEEE Transactions on Multimedia*, v. 20, n. 10, pp. 2836–2848, 2018. doi: 10.1109/TMM.2018.2814339.

ELGELDAWI, E., SAYED, A., GALAL, A. R., et al. "Hyperparameter Tuning for Machine Learning Algorithms Used for Arabic Sentiment Analysis", *Informatics*, v. 8, n. 4, 2021. ISSN: 2227-9709. doi: 10.3390/informatics8040079. Disponível em: <https://www.mdpi.com/2227-9709/8/4/79>.

KARABATAK, M. "A new classifier for breast cancer detection based on Naïve Bayesian", *Measurement*, v. 72, pp. 32–36, 2015. ISSN: 0263-2241. doi: https://doi.org/10.1016/j.measurement.2015.04.028. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0263224115002419>.

REJANI, Y. I. A., SELVI, S. T. "Early Detection of Breast Cancer using SVM Classifier Technique". 2009. Disponível em: <https://arxiv.org/abs/0912.2314>.

ABU-MOSTAFA, Y. S., MAGDON-ISMAIL, M., LIN, H.-T. *Learning From Data: A Short Course*. AMLBook, 2012. ISBN: 9781600490064.

ABU-MOSTAFA, Y. S. "The Learning Problem". In: *Machine Learning Course (CS 156)*, Pasadena, USA, Abr. 2012. California Institute of Technology. Disponível em: <https://youtu.be/mbyG85GZ0PI>.

DUDA, R. O., HART, P. E., STORK, D. G. *Pattern Classification*. 2nd. ed. Califórnia, USA, Wiley, Nov. 2000. ISBN: 978-0-471-05669-0. Disponível em: <https://www.wiley.com/en-us/Pattern+Classification%2C+2nd+Edition-p-9780471056690>.

FERNÁNDEZ-DELGADO, M., CERNADAS, E., BARRO, S., et al. "Do we need hundreds of classifiers to solve real world classification problems?" *Journal of Machine Learning Research*, v. 15, n. 1, pp. 3133–3181, Jan. 2014. ISSN: 1532-4435. doi: 10.5555/2627435.2697065.

AMANCIO, D. R., COMIN, C. H., CASANOVA, D., et al. "A Systematic Comparison of Supervised Classifiers", *PLOS ONE*, v. 9, n. 4, pp. 1–14, Abr. 2014. doi: 10.1371/journal.pone.0094137.

LIM, T.-S., LOH, W.-Y., SHIH, Y.-S. "A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-Three Old and New Classification Algorithms", *Machine Learning*, v. 40, n. 3, pp. 203–228, Set. 2000. ISSN: 1573-0565. doi: 10.1023/A:1007608224229.

SINGH, A., HALGAMUGE, M. N., LAKSHMIGANTHAN, R. "Impact of Different Data Types on Classifier Performance of Random Forest, Naïve Bayes, and K-Nearest Neighbors Algorithms", *International Journal of Advanced Computer Science and Applications*, v. 8, n. 12, 2017. doi: 10.14569/IJACSA.2017.081201.

ZHANG, C., LIU, C., ZHANG, X., et al. "An up-to-date comparison of state-of-the-art classification algorithms", *Expert Systems with Applications*, v. 82, pp. 128–150, 2017. ISSN: 0957-4174. doi: 10.1016/j.eswa.2017.04.003. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0957417417302397>.

KING, R. D., FENG, C., SUTHERLAND, A. "STATLOG: Comparison of Classification Algorithms on Large Real-World", *Applied Artificial Intelligence*, v. 9, n. 3, pp. 289–333, 1995. doi: 10.1080/08839519508945477.

JAPKOWICZ, N., SHAH, M. *Evaluating Learning Algorithms: A Classification Perspective*. Cambridge, GBR, Cambridge University Press, Ago. 2011. ISBN: 9780511921803. doi: 10.1017/CBO9780511921803. Disponível em: <https://www.cambridge.org/core/books/evaluating-learning-algorithms/3CB22D16AB609D1770C24CA2CB5A11BF>.

OH, S. "A new dataset evaluation method based on category overlap", *Computers in Biology and Medicine*, v. 41, n. 2, pp. 115–122, 2011. ISSN: 0010-4825. doi: 10.1016/j.compbiomed.2010.12.006. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0010482510001757>.

HAO, J., HO, T. K. "Machine Learning Made Easy: A Review of 'Scikit-learn' Package in Python Programming Language", *Journal of Educational and Behavioral Statistics*, v. 44, n. 3, pp. 348–361, 2019. ISSN: 1076-9986, 1935-1054. Disponível em: <http://www.jstor.org/stable/45278427>.

CHEN, K., LU, R., WONG, C. K., et al. "Trada: tree based ranking function adaptation". In: *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, p. 1143–1152, Nova York, USA, 2008. Association for Computing Machinery. ISBN: 9781595939913. doi: 10.1145/1458082.1458233.

DOMINGOS, P., PAZZANI, M. "On the Optimality of the Simple Bayesian Classifier under Zero-One Loss", *Machine Learning*, v. 29, n. 2, pp. 103–130, Nov. 1997. ISSN: 1573-0565. doi: 10.1023/A:1007413511361.

FAWAGREH, K., GABER, M. M., ELYAN, E. "Random forests: from early developments to recent advancements", *Systems Science & Control Engineering*, v. 2, n. 1, pp. 602–609, 2014. doi: 10.1080/21642583.2014.956265.

SYARIF, I., PRUGEL-BENNETT, A., WILLS, G. "SVM Parameter Optimization using Grid Search and Genetic Algorithm to Improve Classification Performance", *TELKOMNIKA Telecommunication, Computing, Electronics and Control*, v. 14, n. 4, pp. 1502–1509, Dez. 2016. ISSN: 1693-6930. doi: 10.12928/telkomnika.v14i4.3956. Disponível em: <https://telkomnika.uad.ac.id/index.php/TELKOMNIKA/article/view/3956>.

XIA, Y., LIU, C., LI, Y., et al. "A boosted decision tree approach using Bayesian hyper-parameter optimization for credit scoring", *Expert Systems with Applications*, v. 78, pp. 225–241, 2017. ISSN: 0957-4174. doi: 10.1016/j.eswa.2017.02.017. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0957417417301008>.

HE, H., GARCIA, E. A. "Learning from Imbalanced Data", *IEEE Transactions on Knowledge and Data Engineering*, v. 21, n. 9, pp. 1263–1284, 2009. doi: 10.1109/TKDE.2008.239. Disponível em: <https://ieeexplore.ieee.org/document/5128907>.

FERRI, C., HERNÁNDEZ-ORALLO, J., MODROIU, R. "An experimental comparison of performance measures for classification", *Pattern Recognition Letters*, v. 30, n. 1, pp. 27–38, 2009. ISSN: 0167-8655. doi: 10.1016/j.patrec.2008.08.010. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167865508002687>.

LABATUT, V., CHERIFI, H. "Accuracy Measures for the Comparison of Classifiers". 2012. Disponível em: <https://arxiv.org/abs/1207.3790>.

HIRSCHBERGER, M., QI, Y., STEUER, R. E. "Randomly generating portfolio-selection covariance matrices with specified distributional characteristics", *European Journal of Operational Research*, v. 177, n. 3, pp. 1610–1625, 2007. ISSN: 0377-2217. doi: 10.1016/j.ejor.2005.10.014. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0377221705006600>.

PROBST, P., BOULESTEIX, A.-L. "To Tune or Not to Tune the Number of Trees in Random Forest", *Journal of Machine Learning Research*, v. 18, n. 181, pp. 1–18, 2018. Disponível em: <http://jmlr.org/papers/v18/17-269.html>.

OSHIRO, T. M., PEREZ, P. S., BARANAUSKAS, J. A. "How Many Trees in a Random Forest?" In: Perner, P. (Ed.), *Machine Learning and Data Mining in Pattern Recognition*, pp. 154–168, Berlim, DEU, 2012. Springer Science+Business Media. ISBN: 978-3-642-31537-4. doi: 10.1007/978-3-642-31537-4_13.

LOUPPE, G. "Understanding Random Forests: From Theory to Practice". 2015. Disponível em: <https://arxiv.org/abs/1407.7502>.

INYANG, U. G., IJEBU, F. F., OSANG, F. B., et al. "A Dataset-Driven Parameter Tuning Approach for Enhanced K-Nearest Neighbour Algorithm Performance", *International Journal on Advanced Science, Engineering and Information Technology*, v. 13, n. 1, pp. 380–391, Jan. 2023. doi: 10.18517/ijaseit.13.1.16706. Disponível em: <https://ijaseit.insightsociety.org/index.php/ijaseit/article/view/16706>.

BATISTA, G. E. D. A. P. A., SILVA, D. F. "How k-Nearest Neighbor Parameters Affect its Performance". In: *X Argentine Symposium on Artificial Intelligence*, pp. 1–12, Mar del Plata, ARG, Ago. 2009. Sociedad Argentina de Informática, 38th Jornadas Argentinas de Informática e Investigación Operativa. Disponível em: <https://s.avl.la/6ucxg>.

HANCOCK, J., KHOSHGOFTAAR, T. M. "Impact of Hyperparameter Tuning in Classifying Highly Imbalanced Big Data". In: *2021 IEEE 22nd International Conference on Information Reuse and Integration for Data Science (IRI)*, pp. 348–354, 2021. doi: 10.1109/IRI51335.2021.00054.

CARUANA, R., NICULESCU-MIZIL, A. "Data mining in metric space: an empirical analysis of supervised learning performance criteria". In: *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, p. 69–78, Nova York, USA, 2004. Association for Computing Machinery. ISBN: 1581138881. doi: 10.1145/1014052.1014063.

KOWALSKI, C. J. "On the Effects of Non-Normality on the Distribution of the Sample Product-Moment Correlation Coefficient", *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, v. 21, n. 1, pp. 1–12, 1972. ISSN: 0035-9254, 1467-9876. doi: 10.2307/2346598. Disponível em: <http://www.jstor.org/stable/2346598>.

CHAPELLE, O., VAPNIK, V., BOUSQUET, O., et al. "Choosing Multiple Parameters for Support Vector Machines", *Machine Learning*, v. 46, n. 1, pp. 131–159, Jan. 2002. ISSN: 1573-0565. doi: 10.1023/A:1012450327387.

LAVESSON, N., DAVIDSSON, P. "Quantifying the impact of learning algorithm parameter tuning". In: *Proceedings of the 21st National Conference on Artificial Intelligence*, v. 1, *AAAI'06*, p. 395–400. AAAI Press, 2006. ISBN: 9781577352815. doi: 10.5555/1597538.1597602.

ABU ALFEILAT, H. A., HASSANAT, A. B., LASASSMEH, O., et al. "Effects of Distance Measure Choice on K-Nearest Neighbor Classifier Performance: A Review", *Big Data*, v. 7, n. 4, pp. 221–248, 2019. doi: 10.1089/big.2018.0175. PMID: 31411491.

EL HINDI, K. M., ALJULAIDAN, R. R., ALSALMAN, H. "Lazy fine-tuning algorithms for naïve Bayesian text classification", *Applied Soft Computing*, v. 96, pp. 106652, 2020. ISSN: 1568-4946. doi: 10.1016/j.asoc.2020.106652. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1568494620305901>.

WICKRAMASINGHE, I., KALUTARAGE, H. "Naive Bayes: applications, variations and vulnerabilities: a review of literature with code snippets for implementation", *Soft Computing*, v. 25, n. 3, pp. 2277–2293, Fev. 2021. ISSN: 1433-7479. doi: 10.1007/s00500-020-05297-6.

PROBST, P., WRIGHT, M. N., BOULESTEIX, A.-L. "Hyperparameters and tuning strategies for random forest", *WIREs Data Mining and Knowledge Discovery*, v. 9, n. 3, pp. e1301, 2019. doi: 10.1002/widm.1301.

EL HINDI, K., ALSALMAN, H., QASEM, S., et al. "Building an Ensemble of Fine-Tuned Naive Bayesian Classifiers for Text Classification", *Entropy*, v. 20, n. 11, 2018. ISSN: 1099-4300. doi: 10.3390/e20110857. Disponível em: <https://www.mdpi.com/1099-4300/20/11/857>.

ALONSO, S. G., DE LA TORRE DÍEZ, I., RODRIGUES, J. J. P. C., et al. "A Systematic Review of Techniques and Sources of Big Data in the Healthcare Sector", *Journal of Medical Systems*, v. 41, n. 11, pp. 183, Out. 2017. ISSN: 1573-689X. doi: 10.1007/s10916-017-0832-2.

HOSSAIN, M. E., KABIR, M. A., ZHENG, L., et al. "A systematic review of machine learning techniques for cattle identification: Datasets, methods and future directions", *Artificial Intelligence in Agriculture*, v. 6, pp. 138–155, 2022. ISSN: 2589-7217. doi: 10.1016/j.aiia.2022.09.002. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2589721722000125>.

ANGHEL, A., PAPANDREOU, N., PARNELL, T., et al. "Benchmarking and Optimization of Gradient Boosting Decision Tree Algorithms". 2019. Disponível em: <https://arxiv.org/abs/1809.04559>.

YILMAZ, N., SEKEROGLU, B. "Student Performance Classification Using Artificial Intelligence Techniques". In: Aliev, R. A., Kacprzyk, J., Pedrycz, W., et al. (Eds.), *10th International Conference on Theory and Application of Soft Computing, Computing with Words and Perceptions - ICSCCW-2019*, pp. 596–603, Cham, CHE, 2020. Springer International Publishing. ISBN: 978-3-030-35249-3. doi: 10.1007/978-3-030-35249-3_76.

JENI, L. A., COHN, J. F., DE LA TORRE, F. "Facing Imbalanced Data–Recommendations for the Use of Performance Metrics". In: *2013 Humaine Association Conference on Affective Computing and Intelligent Interaction*, pp. 245–251, Genebra, CHE, Set. 2013. doi: 10.1109/ACII.2013.47. Disponível em: <https://sites.pitt.edu/~jeffcohn/skew/PID2829477.pdf>.

COHEN, J. "A Coefficient of Agreement for Nominal Scales", *Educational and Psychological Measurement*, v. 20, n. 1, pp. 37–46, 1960. doi: 10.1177/001316446002000104.

NGUYEN, V. "Bayesian Optimization for Accelerating Hyper-Parameter Tuning". In: *2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pp. 302–305, Sardenha, ITA, Jun. 2019. ISBN: 978-1-7281-1489-7. doi: 10.1109/AIKE.2019.00060. Disponível em: <https://ieeexplore.ieee.org/document/8791696>.

UNAL, I. "Defining an Optimal Cut-Point Value in ROC Analysis: An Alternative Approach", *Computational and Mathematical Methods in Medicine*, v. 2017, n. 1, Maio 2017. doi: 10.1155/2017/3762651. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1155/2017/3762651>.

Apêndice A

Implementação, em Python, do gerador de dados

```python
# Entradas:
#  N, [1, ∞) ∈ ℕ: Número de pontos no total
#  M, [2, ∞) ∈ ℕ: Número de dimensões/features
#  mean: Correlação entre dimensões (média fora da diagonal)
#  var: Variância (variância fora da diagonal)
def Hirschberger(N, M, mean=0, var=0):
  # Garante parâmetros inteiros
  N = int(N)
  M = int(M)
  # No modo ev − ē, o valor esperado e a variância de elementos da
  # diagonal não são usados. Assim, só precisamos de 'e', 'v' e 'm'
  # § 4., Theorem 1
  # (6) ê = √e/m
  true_mean = np.sqrt(mean / N)
  # (7) v̂ = −ê² + √(ê⁴ + v/m)
  true_var = -true_mean**2 + np.sqrt(true_mean**4 + (var/N))
  # § 6.1. Mode ev − ē
  # (a) Generate standard normally distributed variables q_ij
  Q = np.random.normal(0, 1, M*N).reshape((M, N))
  # (b) Let f_ij = ê + √v̂ q_ij
  F = np.empty((M, N))
  for i in range(M):
    for j in range(N):
      F[i, j] = true_mean + np.sqrt(true_var) * Q[i, j]
  # (c) Compute Σ = FF^⊤
  S = F @ F.T
  # Gera N valores a partir da matriz de covariância, e retorna
  return np.random.multivariate_normal([0]*M, S, N).T
```

Apêndice B

Implementação, em Python, da medida de R-value

```
def Rvalue(X0, X1, K=7, theta=3):
  # R(Ci, Cj) = 1/(|Ci| + |Cj|) * [ r(Ci, Cj) + r(Cj, Ci) ]
  denom = len(X0) + len(X1)
  λ = lambda x: 1 if x > 0 else 0
  def r(Xa, Xb):
    # r(Ci, Cj) = Σ[m=1..Ci] λ(|kNN(p[m], Cj)|-θ)
    s = 0
    for pt in Xa:
      s += λ(countKNNInClass(K, Xb, Xa, pt) - theta)
    return s
  return (1/denom) * (r(X0, X1) + r(X1, X0))

def countKNNInClass(K, Xa, Xb, pt):
  # Encontra os K vizinhos mais próximos de um ponto, em ambas as
   classes
  neighbors = getKNN(K, np.concatenate((Xa, Xb)), pt)[0]
  count = 0
  # Para cada vizinho encontrado
  for ne in neighbors:
    # Se ele está presente na lista de pontos da primeira classe
    if any(np.equal(Xa, ne).all(1)):
      count += 1  # Conta
  # Retorna resultado da contagem
  return count

def getKNN(K, X, target):
  candidates = []
  for current_candidate in X:
    if np.array_equal(current_candidate, target): continue
    # Calcula a distância entre o ponto atual e o objetivo
```

```
30      current_distance = np.linalg.norm(current_candidate - target)
31      # Se ainda não escolhemos K vizinhos
32      if len(candidates) < K:
33        # Então esse está automaticamente selecionado
34        candidates.append((current_candidate, current_distance))
35      # Caso contrário, se já escolhemos K vizinhos
36      else:
37        # Compara o ponto atual com os que já escolhemos (em ordem de
        distância)
38        for i, previous_candidate in enumerate(candidates):
39          # Se a distância for menor do que a de um escolhido
40          if current_distance < previous_candidate[1]:
41            # Adiciona o ponto à lista
42            candidates.append((current_candidate, current_distance))
43            # Ordena por distância
44            candidates.sort(key=lambda el: el[1])
45            # Limita a lista a K pontos
46            candidates = candidates[:K]
47            break
48    out = np.array(candidates, dtype="object")
49    # Separa pontos e distâncias para o retorno
50    return ( np.vstack(out[:,0]).astype("double"), out[:,1].astype("
       double") )
```

Apêndice C

Geração de gráficos de *skew* por métrica de desempenho

```
def generateData(N=1000, skew=1, miss=0):
  # Qtd. de pontos na classe negativa e positiva baseado no skew
  neg = round(N * skew/(1+skew))
  pos = N - neg
  # Cria vetor de pontos com proporção neg/pos
  y_test = np.concatenate((np.zeros(neg), np.ones(pos)), axis=None)

  false_pos = round(neg * miss)
  false_neg = round(pos * miss)
  y_pred = np.concatenate((
    np.zeros(neg - false_pos), np.ones(false_pos),
    np.ones(pos - false_neg),  np.zeros(false_neg)
  ), axis=None)
  return (y_test, y_pred)

def test():
  miss_linspace = [ 0.01, 0.05, 0.1, 0.2, 0.3, 0.5, 0.8, 0.99 ]
  skew_linspace = np.concatenate((np.linspace(1/100, 1, num=20+1),
   np.linspace(5, 100, num=20)))

  metrics = [
    ( "Accuracy", sklearn.metrics.accuracy_score),
    ( "F-score",  sklearn.metrics.f1_score),
    ( "AUC",      sklearn.metrics.roc_auc_score),
    ( "RMSE",     sklearn.metrics.root_mean_squared_error),
    ( "LogLoss",  sklearn.metrics.log_loss),
    ( "MSE",      sklearn.metrics.mean_squared_error),
    ( "Cohen's Kappa", sklearn.metrics.cohen_kappa_score),
    ( "Precision",     sklearn.metrics.precision_score),
    ( "Recall",   sklearn.metrics.recall_score)
```

```
30  ]
31  # Para cada métrica
32  for (name, metric) in metrics:
33    # Uma linha por porcentagem de erro
34    for miss in miss_linspace:
35      # Um datapoint por valor de skew
36      arr = []
37      for skew in skew_linspace:
38        arr.append(metric(*generateData(N=10_000, skew=skew, miss=
     miss)))
39      # Plota a linha
40      plt.plot(skew_linspace, arr, label=f"Miss: {miss*100:.0f}%")
41    plt.title(f"{name} vs. skew")
42    plt.xlabel("Skew")
43    plt.xticks(fontsize=8, rotation=30)
44    plt.xscale("log")
45    plt.ylabel(name)
46    plt.legend(loc="best", fontsize=10)
47    if name == "Cohen's Kappa":
48      plt.ylim([-1.1, 1.1])
49    elif name != "LogLoss":
50      plt.ylim([-0.1, 1.1])
51    plt.show()
```